PATENT ABSTRACTS OF JAPAN

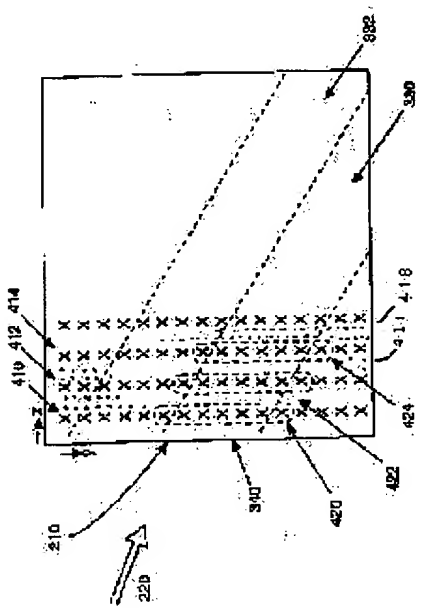
(11)Publication number : **2002-183746**
(43)Date of publication of application : **28.06.2002**

(51)Int.CI. G06T 15/00

(21)Application number : **2000-365865**
(22)Date of filing : **30.11.2000**

(71)Applicant : **TERARECON INC**
(72)Inventor : **LAUER HUGH C**
SEILER LARRY D
GASPARAKIS HARRY
SIMHA VIKRAM
CORRELL KENNETH W

(54) RENDERING METHOD FOR VOLUME DATA SET



(57)Abstract:
PROBLEM TO BE SOLVED: To execute rendering of a volume data set as an image.
SOLUTION: The volume data set includes a large number of voxels stored in a memory. A volume rendering system includes a large number of parallel processing pipelines. The image includes a large number of pixels stored in a memory. A ray set is projected through the volume data set. The volume data set is divided into a large number of sections 330 aligned with the ray set. The voxels along each the ray of each the set is sequentially interpolated only in one of a large number of the pipelines to generate the image only as long as a sample contributes to the image.

(19)日本国特許庁（ＪＰ）　(12) **公開特許公報（Ａ）**　(11)特許出願公開番号
特開2002－183746
（P2002－183746A）
(43)公開日　平成14年６月28日(2002.6.28)

(51)Int.Cl.⁷	識別記号	ＦＩ		ﾃｰﾏｺｰﾄﾞ(参考)
Ｇ０６Ｔ　15/00	２００	Ｇ０６Ｔ　15/00	２００	５Ｂ０８０

審査請求　未請求　請求項の数４　ＯＬ　外国語出願　（全 49 頁）

(21)出願番号　特願2000－365865(P2000－365865)

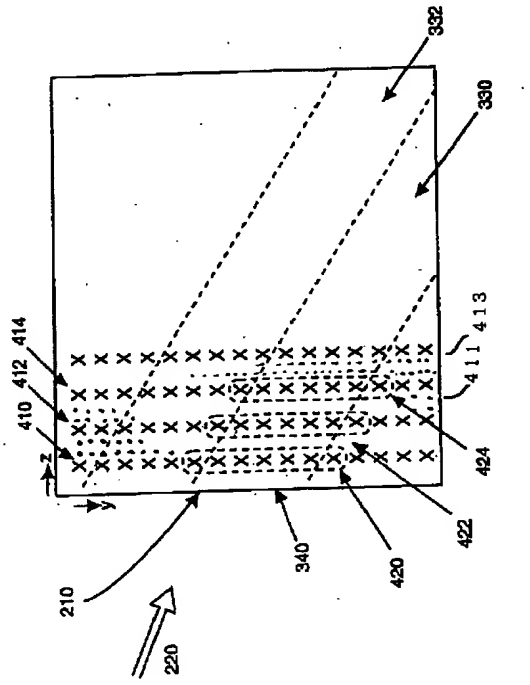
(22)出願日　平成12年11月30日(2000.11.30)

(71)出願人　501184364
テラレコン・インコーポレイテッド
ＴｅｒａＲｅｃｏｎ，　Ｉｎｃ．
アメリカ合衆国、カリフォルニア州、サン・マテオ、キャンパス・ドライブ　2955、スイート　325
(72)発明者　ヒュー・シー・ラウアー
アメリカ合衆国、マサチューセッツ州、コンコード、ボーダー・ロード　69
(74)代理人　100057874
弁理士　曾我　道照　（外７名）

最終頁に続く

(54)【発明の名称】　ボリュームデータ集合のレンダリング方法

(57)【要約】　（修正有）
【課題】ボリュームデータ集合を画像としてレンダリングする。
【解決手段】ボリュームデータ集合は、メモリ中に格納された多数のボクセルを含む。ボリュームレンダリングシステムは、多数の並列処理パイプラインを含む。画像は、メモリ中に格納された多数のピクセルを含む。レイ集合は、ボリュームデータ集合を通して投じられる。ボリュームデータ集合は、レイ集合と整列された多数のセクション３３０に分割される。各集合の各レイに沿ったボクセルは、サンプルが画像に寄与する限りにおいてのみ、画像を発生させるために、多数のパイプラインの１つにおいてのみ逐次補間されたボクセルである。



【特許請求の範囲】
【請求項１】　ボリュームレンダリングシステムにおいてボリュームデータ集合を画像としてレンダリングするための方法であって、前記ボリュームデータ集合はメモリ中に格納された多数のボクセルを含み、前記ボリュームレンダリングシステムは多数の並列処理パイプラインを含み、前記画像は、メモリ中に格納された多数のピクセルを含み、
レイ集合をボリュームデータ集合を通して投じる工程と、
ボリュームデータ集合をレイ集合と整列させた多数のセクションに分割する工程と、
サンプルが画像に寄与する限りにおいてのみ、多数のパイプラインの１つだけの中で各集合の各レイに沿ってボクセルを逐次補間してサンプルを生成させる工程と、
を含むことを特徴とするボリュームデータ集合のレンダリング方法。
【請求項２】　サンプルの不透明度値が予め定めた閾値を超えた時に逐次補間を終了する工程をさらに含むことを特徴とする請求項１に記載のボリュームデータ集合のレンダリング方法。
【請求項３】　終了ビットを集合の各レイに関連付ける工程と、
サンプルの不透明度値が予め定めた閾値を超えた時に終了ビットを設定する工程と、
レイ集合に関連付けられた各終了ビットが設定された時に、レイ集合のための逐次補間を終了する工程と、
をさらに含むことを特徴とする請求項２に記載のボリュームデータ集合のレンダリング方法。
【請求項４】　各セクションについてのマスクが複数のレイの各々についての１つのビットを含み、かつレイの終了時にマスク中にビットを設定する工程と、
前記マスク中のすべてのビットが設定された時に特定のセクションのすべてのレイについての補間を終了する工程と、
をさらに含むことを特徴とする請求項２に記載のボリュームデータ集合のレンダリング方法。
【発明の詳細な説明】
【０００１】
【発明の属する技術分野】本発明は、ボリュームデータ集合のレンダリングに関し、より詳細には、ボリュームデータ集合を通るレイ(光線)投射の早期終了に関する。
【０００２】
【従来の技術】ボリュームグラフィクスは、２次元ないし３次元でサンプリングされたデータ、すなわちボリュームデータ集合として表されたオブジェクトまたはモデルの可視化を扱うコンピュータグラフィクスのサブフィールドである。これらのデータは、ボリュームエレメント、すなわち「ボクセル(voxel)」と呼ばれる。ボクセルは、検討されるオブジェクトまたはモデルの物性を示すデジタル情報を格納する。例えば、特定のオブジェクトまたはモデルのボクセル値は、密度、材質、温度、速度、またはそのオブジェクトまたはモデルの内部または近傍全体にわたる空間中の離散的な点における他の特性を表し得る。
【０００３】ボリュームレンダリングは、印刷、コンピュータ端末上での表示、およびその他の形の可視化のための、２次元画像としてのボリュームデータ集合の投影にかかわるボリュームグラフィクスの分野である。色および透明度の値を特定のボクセルデータ値に割り当てることにより、オブジェクトまたはモデルの外部および内部の種々の眺めがレンダリングされる。
【０００４】例えば、外科手術の事前準備に、患者の膝の靭帯、腱、および骨を検査する必要がある外科医は、膝の断層スキャンを利用し、血液、皮膚、および筋肉に対応するボクセルデータ値を完全に透明にすることができる。その結果得られる画像により、手術前には隠れている靭帯、腱、骨等の状態が明らかになり、それによって、外科手術計画をよりよいものにし、手術時間をより短くし、外科的診察をより少なくし、回復をより早くすることが可能になる。別の例では、ジェットエンジンのタービン翼や溶接継手の断層スキャンを使用する機械工は、固体金属を示すボクセルデータ値を透明にし、他方で空気を示すデータ値を不透明にできる。これにより、そうしなければ人の眼からは隠されている金属内部の欠陥を見ることが可能になる。
【０００５】実時間ボリュームレンダリングは、典型的には毎秒３０フレーム以上の高速で連続的な一連の画像としてのボリュームデータ集合の投影および表示である。これにより、興味あるオブジェクト、モデル、またはシステムの動画を作り出すことが可能になる。さらに、ユーザーに対して視覚的フィードバックを即座に提供する一方で、オペレーターが投影パラメータの制御および画像操作を対話的に行なうことも可能になる。数億個のボクセル値を画像に投影するには膨大な計算能力が必要とされる。実時間でこれを行なうには、実質的により一層の計算能力が要求される。
【０００６】ボリュームレンダリングに関するさらなる背景は、Hanspeter Pfisterが１９９６年１２月にニューヨーク州立大学ストーニーブルック校コンピュータサイエンス学部に提出した博士論文「実時間ボリュームレンダリングのためのアーキテクチャ(Architectures for Real-Time Volume Rendering)」、および米国特許第５，５９４，８４２号「実時間ボリューム可視化用装置および方法(Apparatus and Method for Real-time Volume Visualization)」に含まれている。ボリュームレンダリングに関する付加的な背景は、１９９８年にニュージャージー州アッパーサドルリバーのPrentice Hall PTR発行の、Barthold Lichtenbelt、Randy CraneおよびShaz Naqvi著の「ボリュームレンダリング概論(Introduct

ion to Volume Rendering)」と題された本に示されている。

【０００７】従来技術のボリュームレンダリングパイプライン

従来技術のボリュームレンダリングシステムの１つにおいて、レンダリングパイプラインは、米国特許出願第０９／３１５，７４２号「ボリュームレンダリング集積回路(Volume Rendering Integrated Circuit)」に見られる単一の集積回路チップとして構成されている。

【０００８】「レイキャスティング(ray casting)」と呼ばれる方法においては、レイはボリュームデータ集合を通して投じられ、サンプル点は各レイに沿って計算される。赤、緑、青の色値および不透明度値(アルファ値とも呼ばれる)は、各サンプル点について、サンプル点近傍のボクセルを補間することにより求められる。色値および不透明度値をまとめてＲＧＢＡ値と呼ぶ。これらのＲＧＢＡ値は、典型的には各レイに沿って合成されて最終的なピクセル値を形成し、すべてのレイについてのピクセル値は３次元オブジェクトまたはモデルの２次元画像を形成する。

【０００９】レイキャスティング法に基づいたいくつかのシステムにおいて、１つのレイは画像平面における各ピクセルについてボリュームアレイを通して投じられる。他のシステムにおいては、レイは異なるスペーシングに従って投じられ、次に最終画像は画像平面のピクセル解像度にリサンプリングされる。特に、上記で引用した従来技術のシステムでは、「ビューイング変換のShear-Warp因数分解を使用した高速ボリュームレンダリング(Fast Volume Rendering Using Shear-Warp Factorization of the Viewing Transform)」(Computer Graphics Proceedings of SIGGRAPH, pp. 451-457, 1994)に記載されているLecrouteらの公知のShear-Warpアルゴリズムが使用されている。ここでは、レイはボリュームアレイの１つの面に平行な平面上に等間隔に置かれた点から投じられる。この平面は、ベースプレーンと呼ばれ、点はベースプレーンの軸に整列されている。

【００１０】図１は、米国特許出願第０９／３５３，６７９号「適合性のあるボリュームレンダリングパイプライン(Configurable Volume Rendering Pipeline)」に記載されているような、従来技術のボリュームレンダリングパイプライン１００を表すものである。ボクセルは、ボリュームメモリ１１０から読み取られ、勾配ベクトルを推定するために、勾配推定工程１２０を通される。ボクセルおよび勾配は次に、それらの値をレイに沿ったサンプル点に誘導するために、補間工程１３０を通され、ＲＧＢＡ色および不透明度値を割り当てるために分類工程１４０を通される。その結果得られたＲＧＢＡ値および補間された勾配は、次に照明(イルミネーション)工程１５０へ送られ、ここでハイライトおよびシャドウが加えられる。値は次に、ボリューム部分を除去するため、あるいはそうでない場合はボリュームの画像を変調するために、工程１６０でクリッピングおよびフィルタリングを受ける。最後に、値は工程１７０で合成されて、各レイについてのすべてのＲＧＢＡ値を累積してピクセルメモリ１８０に書き込むための最終的なピクセル値にされる。

【００１１】勾配推定工程１２０、補間工程１３０、および分類工程１４０は、アプリケーションの要件に応じてどのような順序ででも接続できることに注目すべきである。このアーキテクチャは、上記で引用した特許出願に記載されるような、単一の集積回路における並列実行にとりわけ適している。

【００１２】２次元画像合成の代りに、ＲＧＢＡ値を３次元アレイとして書き出し、それによって新規なボリュームデータ集合を作り出すことができ、これは種々の伝達関数によりレンダリングできる。言い換えると、最終画像は、進展型ボリュームデータ集合上の進行性のレンダリングサイクルの結果である。

【００１３】セクションへの分割

単一の半導体集積回路としてボリュームレンダリングエンジンを設計する際の１つの挑戦は、ボリュームレンダリングパイプラインの機能をサポートするのに必要とされるオンチップメモリ量を最小限にすることである。

【００１４】従来技術で実行されるshear-warpアルゴリズムについて図２に示されるように、オンチップメモリ量は、視線方向２２０にほぼ垂直なボリュームデータ集合２００の面２１０の面積に正比例し、これは図２に示されるようにｘｙ面である。従来技術においては、このメモリは、ボリュームをセクションに分割すること、およびボリュームを一度にセクションにレンダリングすることによりレンダリングされた。

【００１５】図２には、ｘ方向およびｙ方向へのボリュームのセクション２３０への分割が示してある。各セクションは、ボリュームアレイのｘｙ面上の面積により定義され、アレイ全体を通してｚ方向へ投影する。これらのセクションは、アレイのｚ軸に平行なので、「軸整列セクション」として知られている。この分割により、オンチップメモリに対する要求が、ボリューム全体の面積でなくてｘｙ平面中のセクションの面の面積に比例する量まで減少する。大きなボリュームデータ集合を一定のオンチップメモリ量で任意にレンダリングできる回路の設計もこれにより可能になる。

【００１６】ボリュームデータ集合を軸整列セクションに分割することの１つの結果は、レイがボリュームを任意角度で横断する際に、レイは１つのセクションから次のセクションへ横切るということである。例えば、視線方向２２０に平行なレイ２４０は、セクション２３１に入り、セクション２３２を横切り、そしてセクション２３３から出る。従って、ボリュームレンダリングパイプラインは、１つのセクションを横断している間に中間記

憶部に蓄積されたレイの部分的合成(中間)値を保たなければならならず、それによりそれらの合成値は、その後のセクションを処理する際に利用できる。さらに、ボリュームレンダリングシステムが並列で動く多数のパイプラインで構成されている場合には、レイの値は、特定のセクション内であっても、１つのパイプラインから次のパイプラインへ渡されなければならない。

【００１７】これら２つの要件により、パイプライン間でのデータのやり取りおよび中間的レイ値の読み書きのためにかなりの量の回路が必要になる。この情報伝達と回路の複雑さを低減することが望ましい。

【００１８】クリッピングおよびクロッピング

ボリュームレンダリングの応用において、オブジェクトの内部が見られるように、切り取りすなわちクリッピングされるボリュームデータ集合の部分を指定することは一般的である(米国特許出願第０９／１９０，６４５号参照)。例えば、カットプレーン(切断面)は、ボリュームアレイ全体を任意の傾斜角でスライスして、オブジェクトの斜めの断面を示すのに使用できる。同様に、クロッププレーン(作付面)の組み合わせは、オブジェクトを特徴的に見せるために使用し得る。

【００１９】ボリュームをクリッピングする別の方法は、奥行(depth)バッファにより表される任意のクリップ表面によるものである(米国特許出願第０９／２１９，０５９号参照)。奥行バッファに関連付けられた奥行テストは、サンプル点を、奥行バッファに関連するそれらの奥行値に基づいて含めるか除外するために用いることができる。

【００２０】図１に示されるような従来技術のパイプラインにおいては、クリッピングおよびクロッピングのためのテストは、合成工程１７０直前の工程１６０で実行された。奥行テストは、合成工程１７０自身の中で行なわれた。その効果は、ボリュームデータ集合のあらゆるボクセルがパイプライン１００内に読み込まれ、すべてのボリュームレンダリングパイプラインではないにしても、レンダリングされているオブジェクトまたはモデルの最終画像にサンプルが寄与しているかどうかに関わらず、ほとんどあらゆるサンプルが最後まで処理された。ほとんどのケースでは、これは不必要な処理および回路の非効率的な利用である。ボクセルおよびサンプルが最終画像に寄与しない場合には、これらを完全にスキップ(飛ばす)することが望ましい。

【００２１】不可視ボクセルおよびサンプルのスキッピング

最終画像においてサンプル点が不可視になるには３通りある。第１に、サンプルはカットプレーン、クロッププレーン、または奥行テストの１つにより最終画像から切り取られ得る。第２に、サンプルは、個別的または集合的に不透明な他のサンプルにより覆い隠され得る。第３に、サンプルは、サンプルが処理されている間に、透明色が割り当てられ得る。

【００２２】明示的にクリッピングされたボクセルまたはサンプルをスキップすることは、「プルーニング(pruning)」と呼ばれる。ボリュームレンダリングのソフトウェア実行においてこれは容易である反面、ハードウェア実行においては困難である。その理由は、従来技術のハードウェアは、従来のダイナミックランダムアクセスメモリ(ＤＲＡＭ)モジュールから所望の性能を得るために、データの系統的あるいは序列的な動きに焦点を当てているからである。そのようなデータをスキップすることにより、ボクセルメモリからボリュームレンダリングパイプラインを通してのデータの規則的な進行が乱され、サンプルおよびそのデータの経過を追うためにコントロール情報の複雑な量が必要となる。

【００２３】ボリュームの他の部分により隠されているサンプルをスキップすることは、「アーリーレイターミネーション(early ray termination：早期レイ終了)」と呼ばれる。また、これはソフトウェア実行においては容易であるが、ハードウェア実行においては非常に困難である。いくつかのレイをスキップし、その他をスキップしないと、メモリからパイプラインを通ってのデータの規則的な流れは、ボクセルおよびサンプルのプルーニングのケースと同様に乱される。

【００２４】アーリーレイターミネーションの１つの形は、Knittelが、「加速されたボリュームレンダリングのための３Ｄテクスチャユニットへのトライアングルキャスタ拡張(Triangle Caster-Extensions to 3D-Texture Units for Accelerated Volume Rendering)」(Proceedings of Eurographics SIGGRAPH, pp. 25-34, 1999)で説明している。ここでは、ボリュームは一連の三角形を３Ｄテクスチャマップを前・後(front-to-back)順に通すことによりレンダリングされる。三角形のすべてのピクセルが不透明であれば、その三角形の背後の三角形はスキップできる。この方法に伴う困難は、この方法がボリュームを直接レンダリングするよりは、最初にボリュームを三角形に変換することに依存する方法であることである。さらに、この方法は、不透明レイをレイベース(ray-by-ray basis)に終了することはできず、三角形ベース(triangle-to-triangle basis)にしかできない。

【００２５】ボリュームの不透明部分をスキップすることは、「スペースリーピング(spaceleaping)」と呼ばれている。一般にこれは、ソフトウェアボリュームレンダリングシステムにおいて行なわれるが、スペースリーピングは、ＲＧＢＡ値を割り当てる伝達関数が変わるたびにボリュームデータ集合を前処理する必要がある。この前処理工程は、実行にコストがかかるが、伝達関数があまり頻繁に変わらないときには、その結果はメリットがある。従来技術のハードウェアシステムにおいては、スペースリーピングも前処理工程を必要とするが、ハードウェア速度でである。しかしながら、スペースリーピン

グを利用すると、メモリからパイプラインを通ってのデータの規則的な流れが、ボクセルおよびサンプルのプルーニングならびにアーリーレイターミネーションの場合と同様に混乱する。

【００２６】Knittelのトライアングル・キャスター・システム(Triangle Caster System)においては、スペースリーピングは、最初にボリュームの非透明部分を、三角形で形成された多面体凸閉包(convex polyhedral hull)で包み込むことにより達成される。閉包外のボリュームの部分は、閉包上での奥行テストにより除外される。１つの問題としては、この方法はホストシステム上でのかなりの量の処理を必要とすることで、この処理は、伝達関数が変化するときはいつでもホストシステムで繰り返されなければならなず、この方法が実時間対話式ボリュームレンダリングに適さないことが考えられる。別の問題としては、レザー式の薄いブレードを備えたタービンのように、大きな陥凹部を有するオブジェクトは、ほとんど凸閉包(convex hull)内部の透明部分で構成されるであろうということである。スペースリーピング用に、レンダリングエンジン自身を用いることが望ましいであろう。

【００２７】

【発明が解決しようとする課題】従って、実時間パフォーマンスで作動するが、画像から切り取られたり、画像の他の部分により隠されたり、あるいは透明であるボクセルおよびサンプルをレンダリングする必要のない、ハードウェアボリュームレンダリングアーキテクチャを有するのが望ましい。

【００２８】

【課題を解決するための手段】この発明は、ボリュームレンダリングシステムにおいて、ボリュームデータ集合を画像としてレンダリングする方法に関する。ボリュームデータ集合は、メモリ中に格納された多数のボクセルを含む。ボリュームレンダリングシステムは、多数の並列処理パイプラインを含む。画像は、メモリ中に格納された多数のピクセルを含む。

【００２９】レイ集合は、ボリュームデータ集合を通して投じられる。ボリュームデータ集合は、レイ集合と整列された多数のセクションに分割される。各集合の各レイに沿ったボクセルは、サンプルが画像に寄与する限りにおいてのみ、画像を生成するために、多数のパイプラインの１つにおいてのみ逐次補間されたボクセルである。

【００３０】

【発明の実施の形態】好ましい実施の形態の詳細な説明

レイ整列セクション

図３に示されるように、本発明のボリュームレンダリングシステムは、ボリュームデータ集合２００をレイ整列セクション３３０～３３２に分割する。各セクションは、ボリュームデータ集合２００のｘｙ面上２１０のエリア３４０によってに定義される。セクションは、従来技術のｚ軸に平行ではなく、選択された視線方向２２０にボリュームを通して投影する。各レイ整列セクションは、従来技術におけるボクセル集合というよりむしろレイの「バンドル(束)」すなわち集合として定義される。

【００３１】各レイ整列セクションは、セクションのレイ集合の各レイについて、最終ＲＧＢＡ値を生成するのに必要とされるすべてのサンプル点を包含する。

【００３２】好ましい実施の形態においては、バンドルを定義するレイのパターンは、典型的には、ボリュームデータ集合２００のｘｙ面２１０上の矩形または平行四辺形である。しかしながら、その他のパターンも選択することができる。例えば、セクション３３０は視線方向２２０に平行なレイのサブセットであり、これはボリュームアレイの面の矩形エリア３４０に当たり、次にｘおよびｙ方向に増大する方向にボリューム中を進む。

【００３３】図4は、ｙｚ平面におけるボリュームデータ集合２００の断面図で、セクション３３０の側面を示したものである。スライスは、ボリュームアレイのｘｙ面に平行なボクセルの２次元アレイである。これらは図４では、「×」記号の垂直カラム４１０、４１２、４１４等として示してある。各スライスの第２の次元はページに垂直である。サンプル４１１および４１３の平面は、ボクセルスライス間の“・”の垂直カラムにより示される。サンプルの密度は、ボクセルのそれとは異なり得る。例えば、ボリュームデータ集合がスーパーサンプリングされると、サンプルの密度はボクセルのそれよりも大きい。ほとんどのケースでは、サンプルはボクセルと合致せず、従ってサンプルは重みを有するコンボルーションカーネル(convolution kernels：畳み込み核)を用いて、近傍のボクセルから補間される。

【００３４】従来技術とは対照的に、本発明のレンダリングシステムは各ボクセルスライスをその前のボクセルスライスから、特定の目視方向についてレイの角度により決まる量でオフセットする。例えば、セクションスライス４２０、４２２および４２４は、セクション３３０をレンダリングするのに必要なスライス４１０、４１２、および４１４のそれぞれサブセットである。セクションスライス４２２は、セクションスライス４２０からｙ方向へオフセットされること、およびセクションスライス４２４は、セクションスライス４２２からｙ方向へオフセットされることに注目されたい。一般に、セクションスライスは、その隣接するスライスからｘおよびｙの両方向へオフセットされ得る。

【００３５】サンプル値の補間およびセクションのエッジに近いサンプルの勾配の推定は、隣接するセクション中のエッジの近傍にあるボクセルに依存することに注目されたい。従って、エッジの近くでは、セクションのボクセルは部分的に重なり合っている。重なりの量は、勾配の推定およびボクセルからのサンプルの補間に必要な

コンボルーションカーネルのサイズにより決まる。
【００３６】レイ整列セクションには多くの利点がある。例えば、レイ全体のサンプルは、以下に説明するように、特定のボリュームレンダリングパイプラインに割り当て得る。結果として、どのような中間的な値も、隣接したパイプライン間を通過させる必要はない。各レイが１つのセクション内に完全に含まれているので、どのようなレイの中間的な値も、１つのセクションから別のセクションへオンチップまたはオフチップメモリを介して伝達される必要がないからである。これらのどちらも、ボリュームレンダリングパイプラインにおいて必要な回路量がかなり減少するという結果になる。
【００３７】レイ整列セクションの他の利点には、サンプルポジションを、従来のグラフィックスエンジンにより生成された奥行バッファと効率的に比較するパイプラインの能力が含まれる。この結果として、任意のボリュームクリッピングおよびレンダリングされたボリュームへのポリゴングラフィックスの埋め込みがどちらも可能になる。レイ整列セクションも、以下でより詳細に説明されるスペースリーピングおよびアーリーレイターミネーションのような多様な最適化を可能にする。
【００３８】ボクセルおよびサンプルのパイプライン処理
レンダリングエンジン構造
図５は、本発明のレンダリングシステムよる、レイ整列されたセクションに分割されたボリュームデータ集合をレンダリングするための集積回路５００および関連メモリモジュール５１０のブロック図である。
【００３９】集積回路５００は、メモリインタフェース５２０，５５０、パイプラインコントローラ９００、複数のボリュームレンダリングパイプライン５４０ａ、．．．、５４０ｄを含んでいる。図５には示してないが、実際のシステムで必要なのは、ホストコンピュータとの通信のためのバスインタフェースである。ユーザーは、メモリ５１０中に格納されたボリュームデータ集合をレンダリングする間、ホストと対話する。
【００４０】集積回路５００は、特定目的ＳＩＭＤ(単一指令多重データ)並列プロセッサとして作動し、制御ロジックを提供するコントローラ９００およびコントローラ指令に呼応して作動する実行エンジンから構成されるボリュームレンダリングパイプライン５４０ａ、．．．、５４０ｄを備えている。各々のパイプラインも、オペレーション中にデータおよびコントロール情報を格納するための多数のレジスターおよびバッファを含んでいる。
【００４１】各パイプライン５４０ａ～５４０ｄは、パイプラインの最初の工程にスライスバッファ５８２、以下、勾配推定ユニット５８４、補間工程５８６、分類工程５８８、照明工程５９０、フィルタ工程５９２、奥行テスト、合成およびアーリーレイターミネーション(ERT)工程５９４、そして奥行および画像バッファ５９６をパイプラインの最後の工程に含んでいる。
【００４２】レンダリングエンジンオペレーション
パイプラインのオペレーション中に、コントローラ９００は多数の機能を実行する。コントローラは、図3のレイ整列されたセクションに従ってボリュームを分割し、レイ整列されたセクションに関連したデータを予め定められた順序で処理する。コントローラは、メモリアクセス指令をメモリインタフェース５５０に送出する。この指令は、メモリ５１０とパイプライン５４０ａ～５４０ｄ間でデータを転送する。例えば、コントローラは、インタフェースに読出し指令を送出し、ボクセルのスライスをメモリ５１０からスライスバッファに分散する。コントローラは奥行バッファ５９６を初期化し、コントローラはピクセル値をメモリ５１０に書き戻す。
【００４３】コントローラは、コンボルーションカーネルが使用する重み５６０も生成する。コンボルーションカーネルは、勾配の推定と補間の間に用いられる。各セクションの完了時、コントローラは、画像をメモリおよび、必要に応じて、更新された奥行バッファに書き込む指令を送出する。
【００４４】各セクションについて、ボリュームオブジェクト内部へのポリゴンオブジェクトの埋め込みを可能にするため、奥行および画像バッファの両方が読取られかつ書き込まれる。あるセクションについて奥行バッファを読取っている間、以下に説明するボクセルプルーニングにおいてコントローラ９００が使用するための、最小および最大(Ｍｉｎ、Ｍａｘ)値５７０が得られる。
【００４５】コントローラ９００は、セクションのレイを複数のパイプライン５４０ａ～５４０ｄの間で均等に割り当てる。処理すべきセクションの準備においてコントローラが奥行および画像バッファを初期化する際、コントローラはレイの分割に従って、奥行および画像バッファ５９６ａ～５９６ｄの間でセクションのデータを分割する。例えば、好ましい実施の形態において、集積回路５００中に４つのボリュームレンダリングパイプラインがあり、それにより各奥行および画像バッファ５９６ａ～５９６ｄはそのセクションについての全奥行および画像値の四分の一を保持する。
【００４６】コントローラ９００によってスライスのボクセルの取り込みが引き起こされる際に、コントローラはそれらをスライスバッファ５８２ａ～５８２ｄに向けて発信し、その結果、各パイプラインはそのレイをレンダリングするのに必要なボクセルを有する。このケースでは、１つのボクセルが典型的には2つ以上のスライスバッファに向けて発信される。なぜなら、特定のボクセルが、多重レイのサンプルの補間に必要となり得るからである。
【００４７】各クロックサイクルにおいて、特定のパイプラインは、勾配の推定および１つのサンプル点の補間

に必要なボクセルをそのスライスバッファ５８２から取得する。このサンプル点は、パイプラインの最後の工程において画像バッファ５９６に合成されるまで、１サイクル毎に１工程ずつパイプラインを通過する。

【００４８】パイプライン方式においては、コントローラ９００により、セクションのすべてのセクションスライスからのボクセルストリームがスライスバッファ５８２ａ～５８２ｄへ発信され、これはパイプライン５４０ａ～５４０ｄがボクセルを処理しかつサンプルを画像バッファ５９６中に合成するのに充分速い。

【００４９】たとえ各セクションスライスが、その近隣のスライスからオフセットされていても、コントローラはセクションスライスを整列させ、その結果、各レイについてのサンプル点は常に同じパイプラインにより処理される。従って、従来技術とは異なり、隣接するパイプライン間の通信は回避される。この結果、集積回路５００がかなり単純化される。

【００５０】スライスバッファ５８２と勾配推定工程５８４の相対的な位置は、スライスバッファに費やされる余剰のオンチップメモリ量を減らすために交換し得る。例えば、ボクセルがスライスバッファに格納される前に、勾配を推定することができる。

【００５１】レイ整列されたセクションのケースでは、ｘおよびｙ方向のスーパーサンプリングは、取るに足らないものになる。なぜならば、コントローラ９００は、ボクセルではなくレイに焦点を当てるからである。スーパーサンプリングを達成するため、もしレイがｘまたはｙ方向にボクセルよりも近くに間隔をおいて配置されるならば、適切なボクセルが、各レイについて責任があるパイプラインに余分に向けられ得る。対照的に、従来技術のパイプラインにおけるボクセルは、レイではなく、パイプラインに関連付けられており、従って、スーパーサンプリングには、パイプラインの間のかなりの通信およびその他の複雑性が要求された。例えば、米国特許願第０９／１９０，７１１２号参照。

【００５２】ボクセルおよびサンプルのプルーニング

本発明のパイプライン式ボリュームレンダリングシステムにおいて、１つの目標は、処理される必要があるボクセルおよびサンプルの数を最小限に抑えることであり、それによって、性能を改善することである。従って、これらのデータを識別するために多くの工程が踏まれる。さらに、別の目標はできる限り早くそのようなデータを識別することである。「可視」および「不可視」データの識別は、コントローラおよびパイプラインの様々な工程によりなされる。

【００５３】例えば、コントローラがセクションを列挙するにつれ、カット平面方程式がテストされ、クロッピング境界がチェックされてセクションの特定のサンプルが視線に含まれているかまたは視線から除外されているかどうかが決定される。もしサンプルが視線から除外されていれば、セクションの処理は、ボクセルを読むことなく、かつボリュームレンダリングパイプラインへサンプルを送ることなく終了し得る。さらに、コントローラは、後に処理されたデータが現在のデータを見えなくし得るかどうかを予期するために「先取り」をする。その場合には、現在のデータの処理はすべて省略し得る。

【００５４】同様に、コントローラは、各セクションについて粗い奥行テストを実行する。コントローラは、アレイがメモリから画像および奥行バッファ５９６ａ～５９６ｄ中へフェッチされるにつれて、各奥行アレイの最小値および最大値５７０を求めることによりこれを行なう。これらの最小値および最大値５７０は、奥行および画像バッファ５９６からコントローラ９００に伝達される。もしコントローラが、セクション内のすべてのサンプルのすべての奥行テストが不合格と決めることができるならば、そのセクションの処理は、ボクセルの読み取りおよびパイプラインへサンプルを送ることなく終了できる。

【００５５】コントローラは、以下に説明されるようにこれらのテストをより細かく繰り返す。例えば、もしコントローラが、特定のスライスまたはスライスのグループのすべてのサンプルがクリップ、クロップ、または奥行テストに失敗することが決定できるなら、これらのスライスはスキップされる。

【００５６】本発明のパイプライン式ボリュームレンダリングシステムにおけるセクションはレイ整列されているので、サンプル、スライス、またはセクション全体のスキッピングはパイプライン５４０ａ～５４０ｄについて何の問題もない。各パイプラインは独立したレイ集合を処理し、各サンプル点は、他のパイプラインのサンプルから独立してパイプラインを通過する。サンプルがパイプラインを流れる際に、各サンプルは「可視性」制御情報を含むことのみが必要であり、その結果、サンプルは最終画像バッファの正しい要素で合成される。もしセクションがボリュームのｚ軸と整列されたら、この制御情報は必然的に非常に複雑なものとなるであろう。

【００５７】従って、各サンプルは、関連した「可視性」ビットを有している。可視性ビットが設定されている限り、サンプルは、処理が有効である。サンプルが最終画像に寄与しないこと、すなわちビットが設定されていないことが決定された時は、サンプルはそれ以上の処理が無効になる。

【００５８】パイプラインを通過させられるサンプルについては、それでも最終奥行テストが実行されなければならないことに注目されたい。なぜなら、そのレイと関連付けられた奥行バッファ値がセクションの最小と最大の間のどこかに位置し得るからである。この最終奥行テストは、工程５９４ａ～５９４ｄにおいて、合成の間に実行される。

【００５９】アーリーレイターミネーション

アーリーレイターミネーションは、レイに沿ったそれ以上のサンプルが、いつ最終的に合成された結果に影響を与えられなくなるかを認識する最適化である。例えば、前・後(front-to-back)処理の間に、完全に不透明な表面に遭遇すると、その表面の背後のサンプルはどれも不可視になり、その結果、これらのサンプルは処理する必要がない。同様に、レイが霧などの厚い半透明な物質を通過する際、結局のところ、レイが不透明度を多く蓄積して、それ以上レイに沿ったサンプルが全く見られなくなり得る。別の例において、レイの奥行値が閾値を超えることがあり、その結果、そのレイに沿ったそれ以上のサンプルはもはや奥行テストを通らない。

【００６０】これらのケースでは、コンポジタ５９４は、レイの処理を終了できることを決め得る。このケースでは、コンポジタは、ＥＲＴ(Early Ray Termination：アーリーレイターミネーション)信号５７２によって、そのレイの処理を停止するようにコントローラに信号を送る。コントローラがあるレイについてそのようなＥＲＴ信号を受け取ると、そのレイに関わるその後のボクセルおよびサンプルをスキップする。コントローラはまた、セクションのすべてのレイについてＥＲＴ信号の経過を追い、その結果、特定のセクション中のすべてのレイが終了される時、セクション全体の処理が終了する。

【００６１】典型的な実施の形態においては、コントローラはビットマスクを維持する。このマスクはセクションの各レイごとに１ビットを有している。マスクのすべてのビットが終了を示す時、そのセクションは終了される。

【００６２】ほとんどのケースにおいて、コンポジタが、あるレイを終了すべきと決めた時に、ボリュームレンダリングパイプラインの前工程にそのレイについてのサンプルがまだあり得ることに注意されたい。従って、アーリーレイターミネーションの意味を維持するため、コンポジタが、あるレイがどのような基準に従っても終了したと決めた後は、そのレイに沿ったその後のすべてのサンプルを無視し、放棄する。このように、１つのレイのアーリー・ターミネーションの時間とそのレイの最後のサンプルがパイプラインを通過し終えるまでの間には遅延があるであろう。

【００６３】好ましい実施の形態において、アーリーレイターミネーションについての不透明度の閾値は、アプリケーションプログラムにより設定できるレジスター中で実現される。従って、たとえ人の眼が、終了されたサンプルの背後のオブジェクトを検知できたとしても、レイを終了するために、どのようなレベルの不透明度も用い得る。

【００６４】スペースリーピング

スペースリーピングにより、パイプラインは、ボリュームの透明部分をスキップすることが可能になる。例えば、多くのボリュームモデルは、対象となるオブジェクトまたはモデルの周囲に１つ以上の透明な「空」スペースを含んでいる。また、分類および照明機能により、ある種の組織または物質を透明にし得る。同様に、勾配の大きさの調節に基づくフィルタリング機能により、ボリュームのある部分を透明にし得る。これらの透明部分をスキップすることにより、不可視サンプルの数が減らされて、従ってボリューム全体をレンダリングするのに必要な時間が減少する。

【００６５】図６および図７に示すように、本発明によるパイプライン式ボリュームレンダリングシステムは、ボリュームアレイ６１０に対応する「空ビット」と呼ばれるビットのアレイ６２０を維持することにより、スペース・リーピングを実現している。三次元のボリュームアレイ６１０については、対応する空ビットの三次元アレイ６２０がある。典型的には、空ビットアレイ中の１つのビット７２０がボクセル７１０の１つのブロックに対応している。もし空ビット７２０が設定されたら、対応するブロック７１０のボクセルは、それらの近くを通過するサンプルの可視性には寄与しないであろう。すなわち、ブロック中のボクセルの各々において、ボリュームは透明である。

【００６６】図７に示すように、各空ビット７２０は、例えば４×４×４ボクセル、すなわち合計６４ボクセルの立方体ブロックに対応し得る。ボクセルの４×４×４の立方体アレイ７１０は、単一の空ビット７２０により表されている。従って、空ビットアレイは、ボクセルの１／６４のビット数を有する。ボクセルが、８、１６、３２、またはそれ以上のビットであり得るので、空ビットアレイは、ボリュームアレイ自身の格納の１／５１２、１／１０２４、１／２０４８、またはそれ以下を必要とする

【００６７】空ビットの意味は次の通りである。サンプル点の補間に必要とされるすべてのボクセルがそれらの空ビットにより透明であると表示されれば、そのサンプル点自身は不可視であり、最終画像に寄与しないと見なされる。従って、コントローラ９００は、それらのボクセルの読み取りを省くことができ、かつボリュームレンダリングパイプラインにサンプル点を送ることを回避できる。

【００６８】多くのボリュームデータ集合において典型的であるように、あるボリュームの大きい部分が、それらの空ビットにより示されるように透明であれば、コントローラは、それらのボクセルの読み取りおよびそれらのサンプルの処理を省き得る。ある領域の空ビットを単に一緒にＡＮＤ演算することにより、コントローラはその領域を効率的にスキップできる。

【００６９】好ましい実施の形態においては、空ビットのアレイ６２０は、特殊モードでの作動、言い換えればホストによる冗長な前処理が全くないモードで、パイプ

ライン自身により生成される。このモードにおいて、視線方向はボリュームの主軸の１つに整列され、サンプル点は、ボクセルポジションと正確に整列するように設定され、それゆえ、補間は些細なことになる。

【００７０】次に、ボリュームは、色および不透明度を割り当てるために所望の伝達関数ならびに勾配量およびその他の因子に基づいた所望のフィルタによって１度レンダリングされる。合成工程５９４ａ～５９４ｄで各ＲＧＢＡサンプルが試験され、サンプルの不透明度が予め定めた閾値未満かどうかを決定する。もし真であれば、そのサンプルおよび対応するボクセルは、透明であると見なされる。もし４×４×４領域中のすべてのボクセルが透明なら、対応するビットは真に設定される。しかしながら、その領域中のいずれかのボクセルが透明でなければ、対応する空ビットは偽に設定される。

【００７１】ボリュームデータ集合の正常なレンダリングの間に、コントローラ９００は、特定のセクションが通過するボリュームの部分についての空ビットのアレイを読取る。もしすべてのビットがボリュームのその部分が透明であることを示せば、ボリュームのその部分のボクセルのスライスは読まれず、これらのスライスに基づくサンプルは処理されない。

【００７２】これは図８に示してある。空ビットの目的のために、ボリュームデータ集合２００は、ボリュームの主軸に整列させた３２×３２×３２のボクセルのブロック８１０に分割してある。上記で説明したように、各ブロックは、空ビットの６４バイトのブロックにより表される。セクション３３０は、視線方向２２０で表される角度でボリュームアレイを通過する。このセクションは、ブロック８１１～８２１と交差する。セクション３３０の処理の一環として、コントローラは、空ビットの対応するブロックを読出し、ビットを一緒にＡＮＤ演算し、対応するボクセルのブロックおよびサンプルをスキップできるかどうかを決定する。セクションがレイと整列されているので、透明なサンプルは、本発明のパイプラインの回路を余分に複雑にすることなくスキップし得る。

【００７３】当然ながら、色および不透明度を割り当てる分類ルックアップテーブルが変わったり、あるいは不透明度に影響するフィルタ機能が変わると、空ビットアレイ６２０は無効になり、再計算されなければならない。しかしながら、これはホストプロセッサ資源を必要としない。

【００７４】パイプラインコントローラ

図９に示すように、本発明のパイプライン式レンダリングシステムのコントローラ９００は以下のように作動する。

【００７５】コントローラには完全なレンダリングコンテクスト９０１が提供される。このレンダリングコンテクストは、ボリュームをレンダリングするために必要なすべての制御情報、例えば視線方向，ボリュームサイズ、カットおよびクロップ平面を定義する方程式、境界ボックス、倍率、奥行平面等を含む。

【００７６】コントローラは、レイ整列セクション９１１に従って、ボリュームデータ集合のレイを分割９１０するためにコンテクストを用いる。セクション９１１は、予め定めた順序に従って処理するために列挙される。各セクションは、ボリュームを通して投じられたそのレイおよびレイに沿ってサンプルを作り出すのに必要なボクセルに関する幾何学という形で表現され得る。

【００７７】各セクション９１１について、工程９２０は上記のような可視性テスト９２０を実行する。視野外(「不可視」)にあるセクションはスキップされ、「可視」セクション９２１のみがさらに処理される。

【００７８】工程９３０は、ボクセルの視点から各可視セクション９２１を処理し、並行して、工程９４０は、サンプルの視点からセクションを処理する。

【００７９】工程９３０は、レイに沿ってスライスの順に命令を実行する。各スライスまたはスライスグループについては、可能であればどこででも、先を見越して、ボクセル可視性テストが上記のように実行される。もしある特定のスライスが少なくとも１つの可視ボクセルを有していれば、コントローラは、スライス読み取り指令命令９３１をメモリインタフェース５５０に送出し、ボクセルのスライス４２０が、メモリ６１０からパイプライン５４０のスライスバッファ５８２の１つに伝達される。もしそのスライスが可視性テストに失敗したら、それはスキップされる。

【００８０】同時に、工程９４０はサンプル可視性テストを実行する。ボクセルおよびサンプルは、たとえ対応するボクセルが通過したとしても、異なる座標系に従って配置されており、従ってサンプルはこのテストに失敗することもあるし、逆もまた同様であることに注目されたい。もしサンプルがパスしたら、サンプル補間指令９４１が補間重み(w)５６０と共に送出され、処理が開始される。その後、パイプラインの様々な工程がさらなる可視性テストを実行し、そのようなアーリーレイターミネーション、奥行テスト、および特定サンプルの処理は中止し得る。

【００８１】ボリュームレンダリング性能における一層の改善のために、付加的な最適化が可能なことが理解されるであろう。例えば、いくつかの実施の形態においては、サンプルスライスは四分円に分割され、各四分円は、カット平面、クロッピング、および奥行テストに従って独立してテストされる。

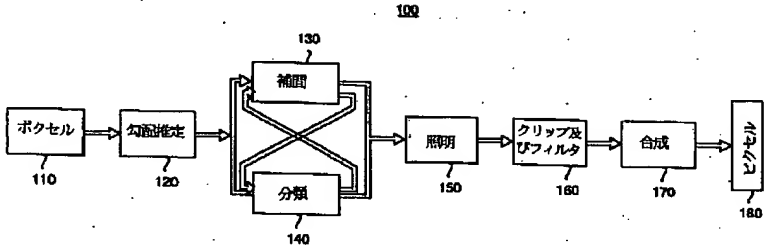
【図面の簡単な説明】

【図１】　従来技術のボリュームレンダリングパイプラインのブロック図である。

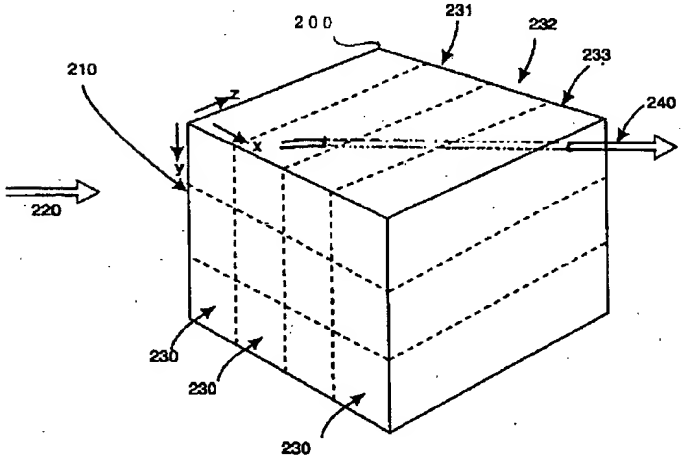
【図２】　従来技術の、軸整列されたセクションへ分割されたボリュームデータ集合の図である。

【図３】　レイ整列されたセクションへ分割されたボリュームデータ集合の図である。
【図４】　レイ整列されたセクションの断面図である。
【図５】　本発明によるボリュームレンダリングパイプラインのブロック図である。
【図６】　ボリュームデータ集合に対応する空ビットの３次元アレイの図である。
【図７】　ボリュームデータ集合のブロックについての空ビットのブロック図である。
【図８】　ボリュームデータ集合の断面図である。
【図９】　図５のレンダリングパイプライン用コントローラの流れ図である。

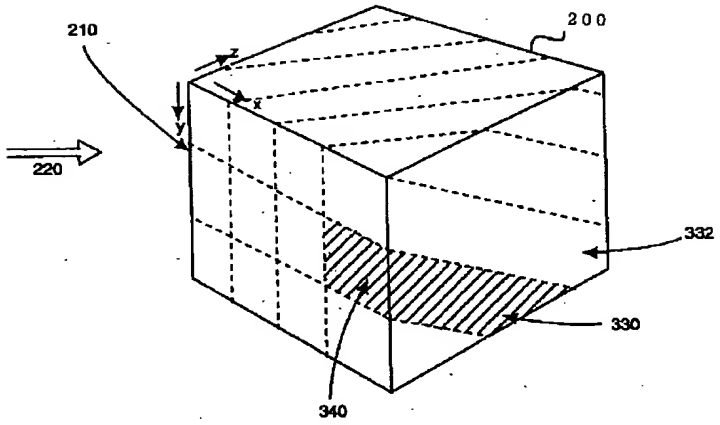
【図１】



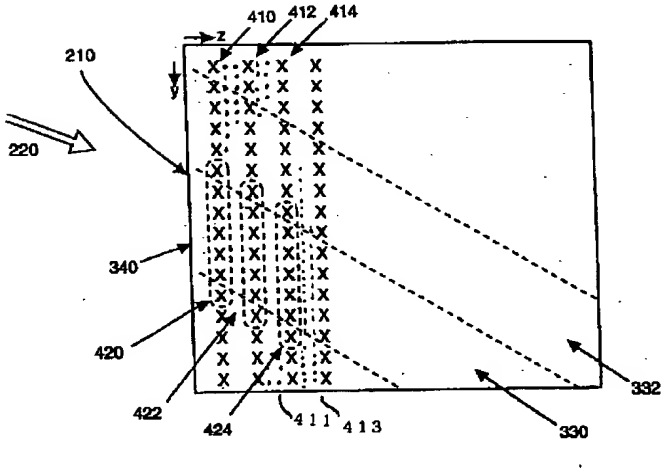
【図２】



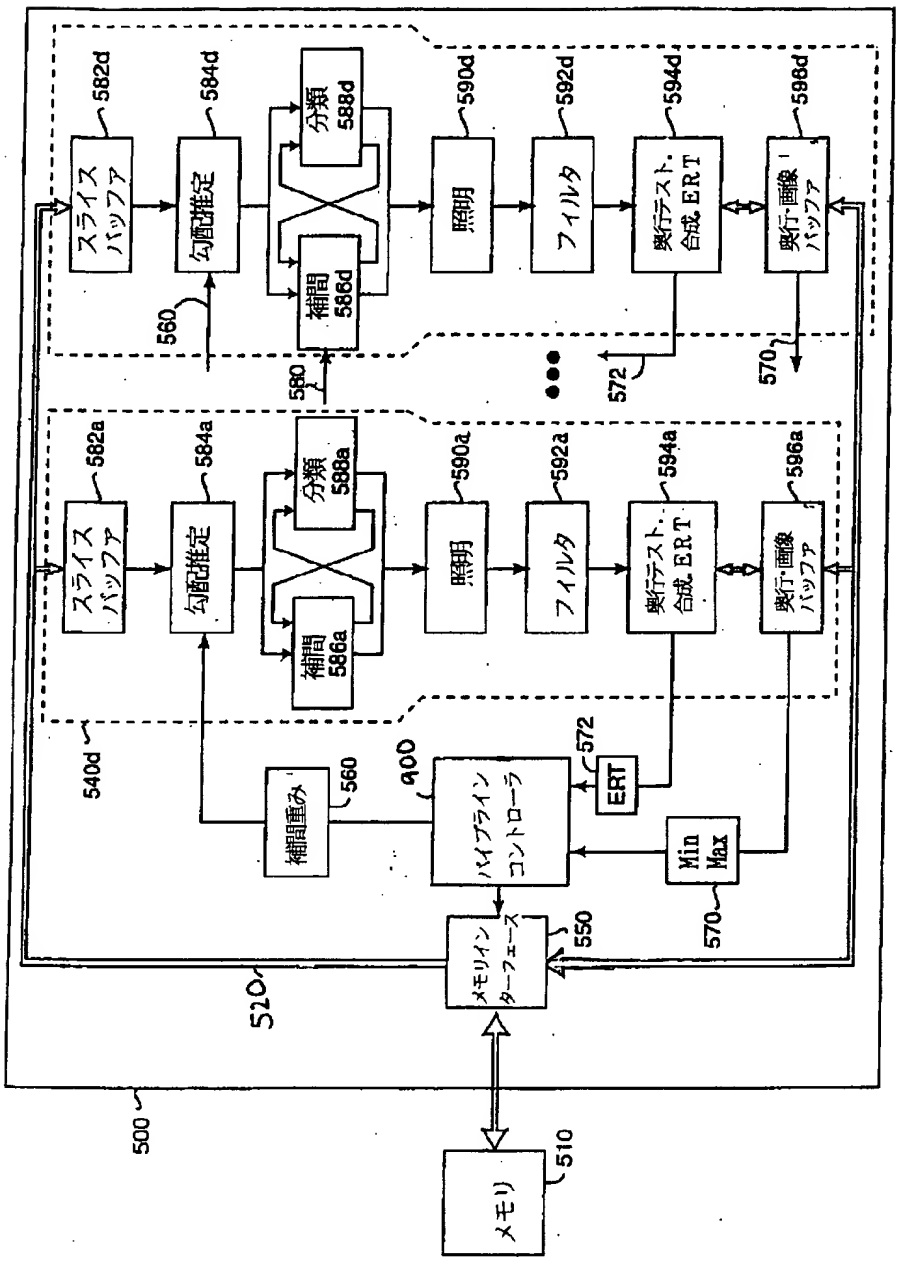
【図3】


210
200
z
x
y
220
332
330
340

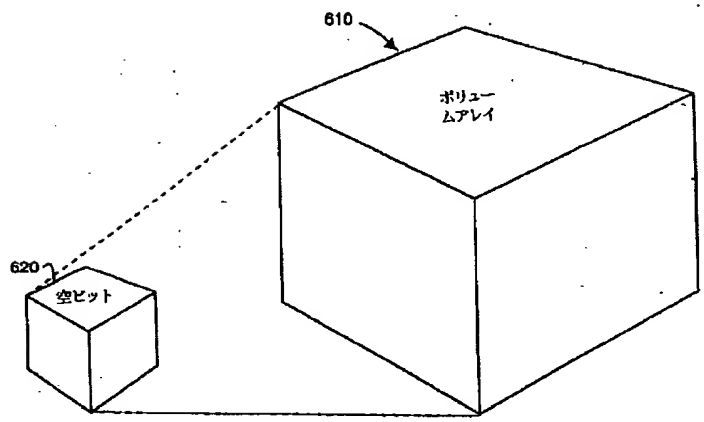
【図4】


410
412
414
z
210
y
220
340
420
422
424
411
413
330
332

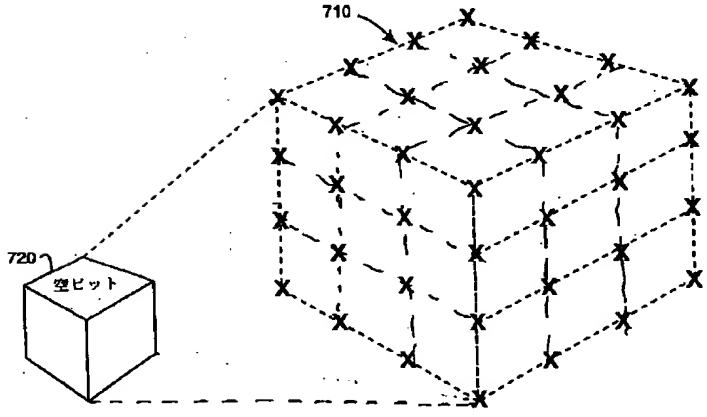
【図5】


582d
584d
分類 588d
590d
592d
594d
596d
スライス バッファ
勾配推定
照明
フィルタ
実行テスト. 合成 ERT
実行 画像 バッファ
560
補間 586d
580
572
570
582a
584a
分類 588a
590a
592a
594a
596a
補間 586a
540d
補間重み
パイプライン コントローラ
900
ERT
Min Max
メモリイン ターフェース
550
520
500
メモリ
510

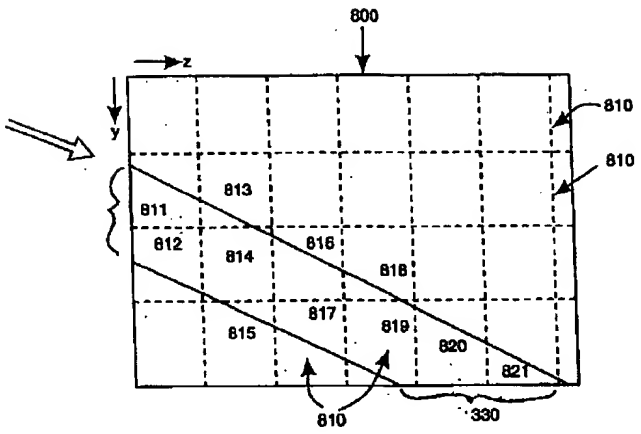
【図6】



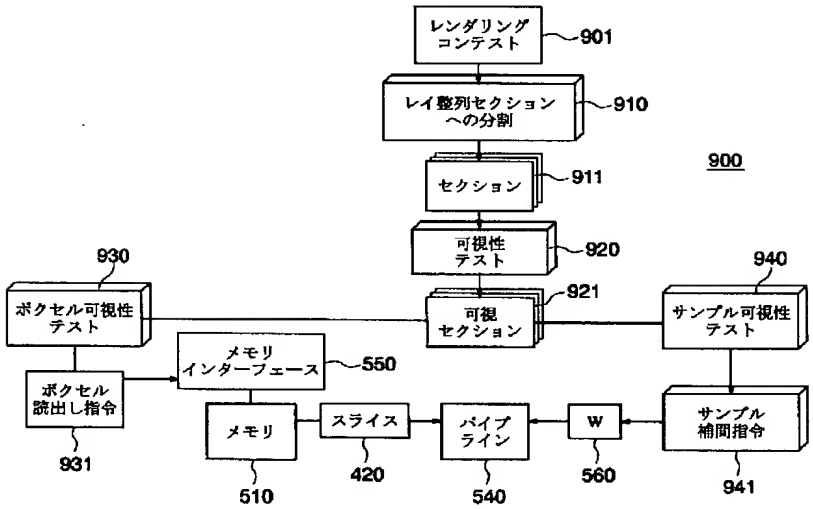
【図7】



【図8】



【図9】



フロントページの続き

(71)出願人 501184364
2955 Campus Drive, Suite 325, San Mateo, California 94403, U.S.A.
(72)発明者 ラリー・ディー・シーラー
アメリカ合衆国、マサチューセッツ州、ボイルストン、リンデン・ストリート 198
(72)発明者 ハリー・ガスパラキス
アメリカ合衆国、マサチューセッツ州、アクトン、デービス・ロード 9、ビー 7
(72)発明者 ヴィックラム・シンハ
アメリカ合衆国、マサチューセッツ州、レキシントン、カターディン・ドライブ 414
(72)発明者 ケニス・ダブリュ・コーラル
アメリカ合衆国、マサチューセッツ州、ランカスター、ルネンバーグ・ロード 2193

Fターム(参考) 5B080 AA17 CA04 FA15 FA17 GA02

【外国語明細書】

1 Title of Invntion
Method for Rendering a Volume Set

2 Claims
1. A method for rendering a volume data set as an image in a volume rendering system, wherein the volume data set includes a plurality of voxels stored in a memory, the volume rendering system includes a plurality of parallel processing pipelines, and the image includes a plurality of pixels stored in the memory; comprising the steps of:

casting sets of rays through the volume data set;

partitioning the volume data set into a plurality of sections aligned with the sets of rays; an

sequentially interpolating voxels along each ray of each set in only one of the plurality of pipelines to generate samples only as long as the samples contribute to the image.

2. The method of claim 1 further comprising the step of:

terminating the sequential interpolating when an opacity value of the samples exceeds a predetermined threshold.

3. The method of claim 2 further comprising the steps of:

associating a termination bit with each ray of the set;

setting the termination bit when the opacity value of the samples exceeds the predetermined threshold; and

terminating the sequential interpolating for the set of rays when each termination bit associated with the set of rays is set.

4. The method of claim 2 wherein a mask for each section includes a bit for each of the plurality of rays, and further comprising the steps of:

setting a bit in the mask when the terminating the ray; and

terminating interpolation for all rays of a particular section when all bits in the mask are set.

3 Detailed Description of Invention

FIELD OF THE INVENTION

This invention relates generally to rendering volume data sets, and more particularly, the invention relates to early termination of rays cast through volume data sets.

BACKGROUND OF THE INVENTION

Volume graphics is the subfield of computer graphics that deals with visualizing objects or models represented as sampled data in three or more dimensions, i.e., a volume data set. These data are called volume elements, or "voxels." The voxels store digital information representing physical characteristics of the objects or models being studied. For example, voxel values for a particular object or model may represent density, type of material, temperature, velocity, or some other property at discrete points in space throughout the interior and in the vicinity of that object or model.

Volume rendering is that part of volume graphics concerned with the projection of the volume data set as two-dimensional images for purposes of printing, display on computer terminals, and other forms of visualization. By assigning color and transparency values to particular voxel data values, different views of the exterior and interior of an object or model can be rendered.

For example, a surgeon needing to examine ligaments, tendons, and bones of a human knee in preparation for surgery can utilize a tomographic scan of the knee and cause voxel data values corresponding to blood, skin, and muscle to appear to be completely transparent. The resulting image then reveals the condition of the ligaments, tendons, bones, etc. which are hidden from view prior to surgery, thereby allowing for better surgical planning, shorter surgical operations, less surgical exploration and faster recoveries. In another example, a mechanic using a tomographic scan of a turbine blade or welded joint in a jet engine can cause voxel data values representing solid metal to appear to be transparent while causing those representing air to be opaque. This allows the viewing of internal flaws in the metal that otherwise is hidden from the human eye.

Real-time volume rendering is the projection and display of the volume data set as a series of images in rapid succession, typically at thirty frames per second or faster. This makes it possible to create the appearance of moving pictures of the object, model, or system of interest. It also enables a human operator to interactively control the parameters of the projection and to manipulate the image, while providing to the user immediate visual feedback. Projecting hundreds of millions of voxel values to an image requires enormous amounts of computing power. Doing so in real time requires substantially more computational power.

Further background on volume rendering is included in a Doctoral Dissertation entitled "Architectures for Real-Time Volume Rendering" submitted by Hanspeter Pfister to the Department of Computer Science at the State University of

New York at Stony Brook in December 1996, and in U.S. Patent No. 5,594,842, "Apparatus and Method for Real-time Volume Visualization." Additional background on volume rendering is presented in a book entitled "Introduction to Volume Rendering" by Barthold Lichtenbelt, Randy Crane, and Shaz Naqvi, published in 1998 by Prentice Hall PTR of Upper Saddle River, New Jersey.

Prior Art Volume Rendering Pipelines

In one prior art volume rendering system, the rendering pipelines are configured as a single integrated chip, U.S. Patent Application Sn. 09/315,742 "Volume Rendering Integrated Circuit."

In a method called "ray-casting," rays are cast through the volume data set, and sample points are calculated along each ray. Red, green, and blue color values, and an opacity value (also called alpha value) is determined for each sample point by interpolating voxels near the sample points. Collectively, the color and opacity values are called RGBA values. These RGBA values are typically composited along each ray to form a final pixel value, and the pixel values for all of the rays form a two-dimensional image of the three-dimensional object or model.

In some systems based on the ray-casting method, one ray is cast through the volume array for each pixel in the image plane. In other systems, rays are cast according to a different spacing, then the final image is resampled to the pixel resolution of the image plane. In particular, the prior art systems cited above uses the well-known Shear-Warp algorithm of Lecroute et al. as described in "Fast Volume

Rendering Using Shear-Warp Factorization of the Viewing Transform," Computer Graphics Proceedings of SIGGRAPH, pp. 451-457, 1994. There, rays are cast from uniformly spaced points on a plane parallel to one of the faces of the volume array. This plane is called the base plane, and the points are aligned with axes of the base plane.

Figure 1 depicts a typical volume rendering pipeline 100 of the prior art, such as described in US Patent Application Sn. 09/353,679 "Configurable Volume Rendering Pipeline." Voxels are read from a volume memory 110 and passed through a gradient estimation stage 120 in order to estimate gradient vectors. The voxels and gradients are then passed through interpolation stages 130 in order to derive their values at sample points along rays, and through classification stages 140 in order to assign RGBA color and opacity values. The resulting RGBA values and interpolated gradients are then passed to illumination stages 150, where highlights and shadows are added. The values are then clipped and filtered in stages 160 in order to remove portions of the volume or otherwise modulate the image of the volume. Finally, the values are composited in stages 170 to accumulate all of the RGBA values for each ray into final pixel values for writing to a pixel memory 180.

It should be noted that the gradient estimation stages 120, interpolation stages 130, and classification stages 140 can be connected in any order, depending upon the requirements of the application. This architecture is particularly suited to parallel implementation in a single integrated circuit, as described in the above referenced Patent Applications.

As an alternative to compositing a two-dimensional image, RGBA values can be written out as a three-dimensional array, thereby creating a new volume data set, which can be rendered with different transfer functions. In other words, the final images is the result of progressive rendering cycles on an evolving volume data set.

Partition into Sections

One of the challenges in designing a volume rendering engine as a single semiconductor integrated circuit is to minimize the amount of on-chip memory required to support the functions of the volume rendering pipelines.

As shown in Figure 2 for the shear-warp algorithm implemented of the prior art, the amount of on-chip memory is directly proportional to an area of a face 210 of a volume data set 200 that is most nearly perpendicular to a view direction 220, that is the *xy*-face as illustrated in Figure 2. In the prior art, this memory was reduced by partitioning the volume into sections, and by rendering the volume a section at a time.

Figure 2 illustrates the partition of the volume into sections 230 in both the *x*- and *y*-directions. Each section is defined by an area on the *xy*-face of the volume array and projects through the entire array in the *z*-direction. These sections are known as "axis-aligned sections" because they are parallel to the *z*-axis of the array. This partition reduces the requirement for on-chip memory to an amount proportional to the area of the face of the section in the *xy*-plane rather than to that of the entire volume. It also makes it possible to design a circuit

capable of rendering arbitrarily large volume data sets with a fixed amount of on-chip memory.

One consequence of partitioning the volume data set into axis-aligned sections is that when rays traverse the volume at arbitrary angles, the rays cross from one section to another. For example, a ray 240 parallel to view direction 220 enters into section 231, crosses section 232, and then exits from section 233. Therefore, the volume rendering pipeline must save the partially composited (intermediate) values of rays that have been accumulated while traversing one section in an intermediate storage, so that those composited values are available when processing a subsequent section. Moreover, when a volume rendering system comprises a number of pipelines operating in parallel, the values of the rays must be passed from one pipeline to the next, even within a particular section.

These two requirements cause the need for a considerable amount of circuitry to communicate data among the pipelines and to write and read intermediate ray values. It is desirable to reduce this communication and circuit complexity.

Clipping and Cropping

It is common in volume rendering applications to specify portions of the volume data set that are cut away or clipped, so that interior portions of the object can be viewed, see U.S. Patent Application Sn. 09/190,645. For example, cut planes can be used to slice through the volume array at any oblique angle, showing an angled cross-section of the object.

Similarly, combinations of crop planes may be employed to provide distinctive views of an object.

Another method of clipping a volume is by an arbitrary clip surface represented by a depth buffer, see U.S. Patent Application Sn. 09/219,059. Depth tests associated with the depth buffer can be used to include or exclude sample points based on their depth values relative to the depth buffer.

In a prior art pipeline, such as represented by Figure 1, tests for clipping and cropping were implemented in stages 160 just prior to the compositor 170. Depth tests were implemented in the compositor 170 itself. The effect was that every voxel of the volume data set was read into the pipeline 100, and every sample was processed through most, if not all of the volume rendering pipeline, whether or not the sample contributed to the final image of the object or model being rendered. In most cases, this represents unnecessary processing and inefficient use of circuitry. It is desirable to skip over voxels and samples completely if those voxels do not contribute to the final image.

Skipping Non-Visible Voxels and Samples

There are three ways that a sample point may not be visible in the final image. First, the sample may be clipped out of the final image by one of the cut planes, crop planes, or depth tests. Second, the sample may be obscured by other samples which are individually or collectively opaque. Third, the sample may have been assigned a transparent color while the sample is processed.

Skipping over voxels or samples that are explicitly clipped is called "pruning." While this is easy in software implementations of volume rendering, pruning is difficult in hardware implementations. The reason is that prior art hardware systems focus on the systematic movement of data in order to obtain the desired performance from conventional dynamic random access memory (DRAM) modules. By skipping over such data, the orderly progression of data from voxel memory through the volume rendering pipelines is upset, and a complex amount of control information is necessary to keep track of samples and their data.

Skipping samples that are obscured by other parts of the volume is called "early ray termination." Again, this is easy in software implementations, but very difficult in hardware implementations. Skipping some rays and not others also upsets the orderly progression of data flowing from the memory through the pipelines, just as in the case of pruning voxels and samples.

One form of early ray termination is described by Knittel in "TriangleCaster - Extensions to 3D-Texture Units for Accelerated Volume Rendering," Proceedings of Eurographics SIGGRAPH, pp. 25-34, 1999. There, volumes are rendered by passing a series of triangles through a 3D texture map in a front-to-back order. If all of the pixels of a triangle are opaque, then triangles behind that triangle can be skipped. The difficulty with that method is that the method depends on first converting the volume to triangles, rather than rendering the volume directly. Moreover, that method cannot terminate opaque rays on a ray-by-ray basis, only on a triangle-to-triangle basis.

Skipping over transparent parts of the volume is called "space leaping." This is commonly practiced in software volume rendering systems, but space leaping requires that the volume data set is pre-processed each time transfer functions assigning RGBA values change. This pre-processing step is costly in performance, but the result is a net benefit when the transfer functions do not change very often. In prior art hardware systems, space leaping also required a pre-processing step, but at hardware speeds. However, taking advantage of space-leaping upsets the ordered progression of data flowing from the memory through the pipelines, just as pruning voxels and samples and just as early ray termination.

In Knittel's TriangleCaster system, space leaping is accomplished by first encapsulating non-transparent portions of the volume with a convex polyhedral hull formed of triangles. Portions of the volume outside the hull are excluded by using depth tests on the hull. As a problem, that method requires a significant amount of processing on the host system which must be repeated whenever the transfer function changes, possible making that method unsuitable for real-time interactive volume rendering. As another problem, an object with large concavities, such as a turbine with razor thin blades, will mostly consist of transparent portions within the convex hull. It would be desirable to use the rendering engine itself for space leaping.

Therefore, it is desirable to have a hardware volume rendering architecture that operates with real-time performance but does not have to render voxels and samples that are clipped out of the image, that are obscured by other parts of the image, or that are transparent.

SUMMARY OF THE INVENTION

A method renders a volume data set as an image in a volume rendering system. The volume data set includes a plurality of voxels stored in a memory. The volume rendering system includes a plurality of parallel processing pipelines. The image includes a plurality of pixels stored in the memory.

A set of rays are cast through the volume data set. The volume data set is partitioned into a plurality of sections aligned with the sets of rays. Voxels along each ray of each set are sequentially interpolated voxels in only one of the plurality of pipelines to generate samples only as long as the samples contribute to the image.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

Ray-aligned sections

As shown in Figure 3, the present volume rendering system partitions the volume data set 200 into *ray-aligned* sections 330-332. Each section is defined by an area 340 on an *xy*-face 210 of a volume data set 200. The sections project through the volume in a direction parallel to a selected view direction 220 rather than parallel to the *z*-axis as in the prior art. Each ray-aligned section is defined as a "bundle" or set of rays, rather than a set of voxels as in the prior art.

Each ray-aligned section encompasses all of sample points needed to produce a final RGBA pixel values for each ray of the section's ray set.

In the preferred embodiment, the pattern of the rays defining the bundle is typically a rectangle or parallelogram on the *xy*-face 210 of the volume data set 200. However, other patterns can also be chosen. For example, section 330 is the subset of rays parallel to the view direction 220 that strike the rectangular area 340 of the face of the volume array, then proceed through the volume in an increasing *x*- and *y*-direction.

Figure 4 is a cross-sectional view of the volume data set 200 in the *yz*-plane with a side view of section 330. Slices are two-dimensional arrays of voxels parallel to the *xy*-face of the volume array. These are depicted in Figure 4 as vertical columns of "×" symbols 410, 412, 414, etc. The second dimension of each slice is perpendicular to the page. Planes of samples 411 and 413 are shown by vertical columns of "·" between the voxel slices. The density of the samples can be different than that of the voxels. For example, if the volume data set is supersampled, the density of the samples is greater than that of the voxels. In most instances, the samples will not coincide with the voxels, therefore, the samples are interpolated from nearby voxels using convolution kernels having weights.

In contrast to the prior art, the present rendering system offsets each voxel slice from the previous voxel slice by an amount determined by the angle of the ray for a particular view direction. For example, section slices 420, 422, and 424 are subsets respectively of slices 410, 412, and 414 needed to render section 330. Note that section slice 422 is offset in the *y*-direction from section slice 420 and that section slice 424 is offset in the *y*-direction from section

slice 422. In general, a section slice may be offset in both the *x*- and *y*-directions from its neighbor.

Note that interpolation of sample values and estimation of gradients of samples near the edge of the section depend upon voxels that lie near the edges in adjacent sections. Therefore, near the edges, the voxels of the sections partially overlap. The amount of overlap is determined by the size of the convolution kernel needed for estimating gradients and interpolating the samples from the voxels.

Ray-aligned sections bring a number of advantages. For example, the samples of an entire ray may be assigned to a particular volume rendering pipeline, as described below. As a result, no intermediate values need to be passed among adjacent pipelines. Because each ray is contained entirely within one section, no intermediate values of rays need be communicated from one section to another via on-chip or off-chip memory. Both of these result in considerable reductions in the amount of circuitry needed in the volume rendering pipelines.

Other advantages of ray-aligned sections include the ability of the pipeline to efficiently compare sample positions to depth buffers produced by a traditional graphics engine. This in turn enables both arbitrary volume clipping and embedding polygon graphics into the rendered volume. Ray-aligned sections also allow a variety of optimizations, such as space leaping and early ray termination described in greater detail below.

Pipelined Processing of Voxels and Samples

Rendering Engine Structure

Figure 5 is a block diagram of an integrated circuit 500 and associated memory modules 510 for rendering a volume data set partitioned into ray-aligned sections according to the present rendering system.

The integrated circuit 500 includes a memory interface 520, a pipeline controller 900, and a plurality of volume rendering pipelines 540a, ..., 540d. Not shown in Figure 5, but necessary in a practical system, is a bus interface for communication with a host computer. A user interacts with the host while rendering the volume data set stored in the memory 510.

The integrated circuit 500 acts as a special purpose SIMD (Single-Instruction, Multiple Data) parallel processor, with the controller 900 providing control logic and the volume rendering pipelines 540a, ..., 540d comprising the execution engines operating in response to controller commands. Each of the pipelines also includes numerous registers and buffers for storing data and control information during operation.

Each pipeline 540a-d includes a slice buffer 582 at a first stage of the pipeline, a gradient estimation unit 584, interpolation stages 586, classification stages 588, illumination stages 590, filter stages 592, depth test, composite and early ray termination (ERT) stages 594, and depth and image buffers 596 at a last stage of the pipeline.

Rendering Engine Operation

During operation of the pipelines, the controller 900 performs a number of functions. The controller partitions the volume according to the ray-aligned sections of Figure 3, and processes data associated with the ray-aligned sections in a predetermined order. The controller issues memory access commands to the memory interface 550. The commands transfer data between the memory 510 and the pipelines 540a-d. For example, the controller issues read commands to the interface to distribute slices of voxels from the memory 510 to the slice buffers. The controller initializes the depth buffers 596, and the controller writes pixel values back to the memory 510.

The controller also generates the weights 560 used by the convolution kernels. The convolution kernels are used during gradient estimation and interpolation. At the completion of each section, the controller issues commands to write images to the memory, and to updated depth buffers as necessary.

For each section, depth and image buffers are both read and written in order to allow embedding of polygon objects within images of volume objects. While reading depth buffers for a section, minimum and maximum (Min, Max) values 570 are obtained for use by controller 900 in voxel pruning described below.

The controller 900 assigns the rays of a section equally among the plurality of pipelines 540a, ..., 540d. When the controller initializes the depth and image buffers in preparation a section to be processed, the controller partitions section's data among the depth and image buffers 596a, ..., 596d according to the partitioning of the rays. For

example, in the preferred embodiment, there are four volume rendering pipelines in integrated circuit 500, so that each depth and image buffer 596a, ..., 596d holds one fourth of the total depth and image values for the section.

When the controller 900 causes voxels of a slice to be fetched, the controller directs them to the slice buffers 582a, ..., 582d so that each pipeline has the necessary voxels to render its rays. In this case, a voxel will typically be directed to more than one slice buffer because a particular voxel may be needed for interpolating samples of multiple rays.

In each clock cycle, a particular pipeline takes voxels needed for gradient estimation and interpolation of one sample point from its slice buffer 582. This sample point then passes down the pipeline, one stage per cycle, until it is composited into the image buffer 596 at the last stage of the pipeline.

In pipeline fashion, the controller 900 causes a stream of voxels from all the section slices of a section to be directed to the slice buffers 582a, ..., 582d, fast enough for the pipelines 540a, ..., 540d to process the voxels and composite samples into the image buffer 596.

Even though each section slice is offset from its neighboring slice, the controller aligns the section slices so that sample points for each ray are always processed by the same pipeline. Therefore, unlike the prior art, communication among adjacent pipelines is avoided. This results in a considerable simplification of the integrated circuit 500.

The relative position of the slice buffers 582 and the gradient estimation stage 584 may be interchanged in order to reduce

the amount of redundant on-chip memory devoted to the slice buffers. For example, gradients can be estimated before the voxels are stored in the slice buffers.

Supersampling in the *x*- and *y*-directions becomes trivial with ray-aligned sections because the controller 900 focuses on rays, not voxels. If rays are spaced closer than voxels in either the *x*- or the - or the *y*-direction to achieve supersampling, then appropriate voxels can be directed redundantly to the pipelines responsible for the respective rays. In contrast, voxels in prior art pipelines were associated with pipelines, not rays, and therefore considerable communication among pipelines and other complexity were required for supersampling, see for example, U.S. Patent Application 09/190,712

Voxel and Sample Pruning

In the present pipelined volume rendering system, one goal is to minimize the number of voxels and samples that need to be processed, thereby improving performance. Therefore, numerous steps are taken to identify these data. Furthermore, another goal is to identify such data as early as possible. The identification of "visible" and "invisible" data is done both by the controller and the various stages of the pipelines.

For example, as the controller enumerates sections, cut plane equations are tested, and cropping boundaries are checked to determine whether particular samples of a section are included or excluded from the view. If the samples are excluded from the view, then the processing of the section can be terminated without reading voxels and without sending samples down the volume rendering pipelines.

Furthermore, the controller "looks-ahead" to anticipate whether later processed data can cause current data to become invisible, in which case processing of current data can be skipped altogether.

Likewise, the controller performs coarse grain depth tests for each section. The controller does this by determining the minimum and maximum values 570 of each depth array as the array is fetched from memory into the image and depth buffers 596a, ..., 596d. These minimum and maximum values 570 are communicated to the controller 900 from the depth and image buffers 596. If the controller can determine that all of the depth tests of all of the samples within a section fail, then processing of that section can be terminated without reading voxels and without sending samples down the pipelines.

The controller repeats these tests at a finer grain as described below. For example, if the controller can be determined that all of the samples of a particular slice or group of slices fail a clip, crop, or depth test, then those slices are skipped.

Because sections in the present pipelined volume rendering system are ray aligned, the skipping of samples, slices, or whole sections presents no problem for the pipelines 540a, ..., 540d. Each pipeline processes an independent set of rays and each sample point passes down the pipeline independent of samples in other pipelines. It is only necessary to include "visibility" control information for each sample as the sample flows down the pipeline, so that the sample is composited with the correct element of the final image buffer. If sections were aligned with the *z*-axis of the volume, then this control information would entail great complexity.

Therefore, each sample has an associated "visibility" bit. As long as the
visibility bit is set, the sample is valid for processing. When it is determined that the sample will not contribute to the final image, that is the bit is unset, and the sample becomes invalid for further processing.

Note that for samples that are passed down the pipelines, a final depth test must still be performed because the depth buffer value associated with that ray may lie somewhere between the minimum and maximum of the section. This final depth test is performed during compositing, in stage 594a, ..., 594d.

Early Ray Termination

Early ray termination is an optimization that recognizes when further samples along a ray cannot affect the final composited result. For example, during front-to-back processing, when a fully opaque surface is encountered, no samples behind the surface will be visible, and hence these samples need not be processed. Similarly, when rays pass through thick translucent material such as fog, they may eventually accumulate so much opacity that no samples further along the ray can be seen. In another example, the depth values of a ray may cross a threshold so that no further samples along that ray pass depth tests.

In these cases, the compositor 594 may determine that processing of the ray can be terminated. In this case, the compositer signals the controller to stop processing that ray via an ERT (Early Ray Termination) signals 572. When the

controller receives such the ERT signal for a ray, it skips over subsequent voxels and samples involving that ray. The controller also keeps track of the ERT signals for all of the rays of a section, so that when all rays in a particular section are terminated, processing for the entire section is terminated.

In a typical implementation, the controller maintains a bit mask. The mask has one bit for each ray of the section. When all of the bits of the mask indicate termination, then the section is terminated.

Note that in most cases, when the compositor has determined that a ray should be terminated, there may still be samples for that ray in previous stages of the volume rendering pipeline. Therefore, in order to preserve the semantics of early ray termination, the compositer ignores and discards all subsequent samples along that ray after the compositor has determined that a ray has terminated according to any criterion. Thus, there will be a delay between the time of early termination of one ray and the last sample of that ray to complete its course through the pipeline.

In the preferred embodiment, the threshold of opacity for early ray termination is implemented in a register that can be set by an application program. Therefore, any level of opacity may be used to terminate rays, even if the human eye were capable of detecting objects behind the terminated sample.

Space Leaping

Space leaping allows the pipelines to skip over transparent portions of the volume. For example, many volume models

include one or more portions of transparent "empty" space around the object or model of interest. Also, classification and lighting functions may make certain types of tissue or material transparent. Similarly, filtering functions based on modulation of gradient magnitudes can cause portions of the volume to become transparent. By skipping these transparent portions, the number of invisible samples is reduced, thereby reducing the time needed to render the entire volume.

As shown in Figures 6 and 7, the present pipelined volume rendering system implements space leaping by maintaining an array of bits 620 called "empty bits" corresponding to the volume array 610. For the three dimensional volume array 610, there is a corresponding three dimensional array of empty bits 620. Typically, one bit 720 in the array of empty bits corresponds to a block of voxels 710. If the empty bit 720 is set, then the voxels of the corresponding block 710 will not contribute to the visibility of samples passing near them. That is, the volume at each of the voxels in the block is transparent.

As shown in Figure 7, each empty bit 720 can correspond to a cubic block of, for example, 4×4×4 voxels, or 64 voxels in total. The 4×4×4 cubic array of voxels 710 is represented by the single empty bit 720. Therefore, the empty bit array has $1/64^{th}$ the number of bits as there are voxels. Because voxels may be 8, 16, 32, or more bits, the empty bit array requires $1/512^{th}$, $1/1024^{th}$, $1/2048^{th}$, or a smaller fraction of the storage of the volume array itself.

The semantics of empty bits are as follows. If all of the voxels needed to interpolate a sample point are indicated as transparent by their empty bits, then the sample point itself is

deemed to be invisible and not contributing to the final image. Therefore, the controller 900 can omit the reading of those voxels, and can avoid sending the sample points down the volume rendering pipeline.

If, as is typical in many volume data sets, a large portion of a volume is transparent as indicated by their empty bits, then the controller can omit reading those voxels and processing those samples. By simply ANDing together the empty bits of a region, the controller can efficiently skip the region.

In the preferred embodiment, the array of empty bits 620 is generated by the pipelines themselves, operating in a special mode, in other words, there is no lengthy preprocessing by the host. In this mode, the view direction is aligned with one of the major axes of the volume, and the sample points are set to be exactly aligned with voxel positions, hence interpolation becomes trivial.

Then, the volume is rendered once with desired transfer functions to assign color and opacity and with desired filters based on gradient magnitude and other factors. The compositing stages 594a, ..., 594d examine each RGBA sample to determine whether the opacity of the sample is less than a predetermined threshold. If true, then the sample and its corresponding voxel are deemed to be transparent. If all of the voxels in a 4×4×4 region are transparent, then the corresponding empty bit is set to true. If, however, any voxel in the region is not transparent, then the corresponding empty bit is set to false.

During normal rendering of a volume data set, the controller 900 reads a array of empty bits for the parts of the volume

through which a particular section passes. If all of the bits indicate that the portion of the volume is transparent, then the slices of voxels in that portion of the volume are not read and the samples based on these slices are not processed.

This is illustrated in Figure 8. For the purpose of the empty bits, the volume data set 200 is partitioned into 32×32×32 blocks of voxels 810 aligned with the major axes of the volume. Each block is represented by a 64-byte block of empty bits as described above. Section 330 passes through the volume array at an angle represented by view direction 220. The section intersects with blocks 811-821. As part of processing the section 330, the controller fetches the corresponding blocks of empty bits, ANDs the bits together and determines whether or not the corresponding blocks of voxels and samples can be skipped. Because sections are aligned with rays, transparent samples may be skipped without adding extra complexity to the circuitry of the present pipeline.

Naturally, if classification lookup tables assigning color and opacity change, or if filter functions affecting modulation of opacity change, then the empty bit array 620 becomes invalid, and must be recomputed. However, this does not require host processor resources.

Pipeline Controller

As illustrated by Figure 9, the controller 900 of the present pipelined rendering system operates as follows.

The controller is provided with a complete rendering context 901. The rendering context includes all control information

needed to render the volume, for example, the view direction, the size of the volume, equations defining cut and crop planes, bounding boxes, scaling factors, depth planes, etc.

The controller uses the context to partition 910 the rays of the volume data set according to ray aligned sections 911. The sections 911 are enumerated for processing according to a predetermined order. Each section can be expressed in terms of its geometry with respect to its rays cast through the volume, and with respect to voxels needed to produce samples along the rays.

For each section 911, step 920 performs visibility test 920 as described above. Sections that are outside the field of view ("invisible") are skipped, and only "visible" sections 921 are further processed.

Step 930 processes each visible section 921 from the perspective of voxels, and in parallel, step 940 processes the section from the perspective of samples.

Step 930 steps along rays in slice order. For each slice, or group of slices, with look-ahead wherever possible, voxel visibility tests are performed as described above. If a particular slice has at least one visible voxel, then the controller issues a read slice command 931 to the memory interface 550, and a slice of voxels 420 is transferred from the memory 610 to one of the slice buffers 582 of the pipelines 540. If the slice fails the visibility test, then it is skipped.

Concurrently, step 940 performs sample visibility tests. Note, that voxels and samples are arranged according different

coordinate systems, therefore samples might fail this test, even though corresponding voxels passed, and vice versa. If the sample passes, then an interpolate sample command 941 is issued, along with interpolation weights (w) 560, and processing commences. Subsequently, the various stages of the pipeline perform further visibility tests, such early ray termination, and depth testing, and processing of a particular sample can be aborted.

It will be appreciated that additional optimizations for further improvements in volume rendering performance. For example, in certain embodiments, sample slices are partitioned into quadrants, with each quadrant being tested independently for visibility according to the cut plane, cropping, and depth tests.

4 Brief Description of Drawings

Figure 1 is a block diagram of a prior art volume rendering pipeline;

Figure 2 is a diagram of a volume data set partitioned into axis-aligned sections of the prior art;

Figure 3 is a diagram of a volume data set partitioned into ray-aligned sections;

Figure 4 is a cross-sectional view of a ray-aligned section;

Figure 5 is a block diagram of volume rendering pipelines according to the invention;

Figure 6 is a diagram of a three-dimensional array of empty bits corresponding to the volume data set;

Figure 7 is a block diagram of one empty bit for a block of the volume data set;

Figure 8 is a cross-section of the volume data set; and

Figure 9 is a flow diagram of a controller for the rendering pipelines of Figure 5.

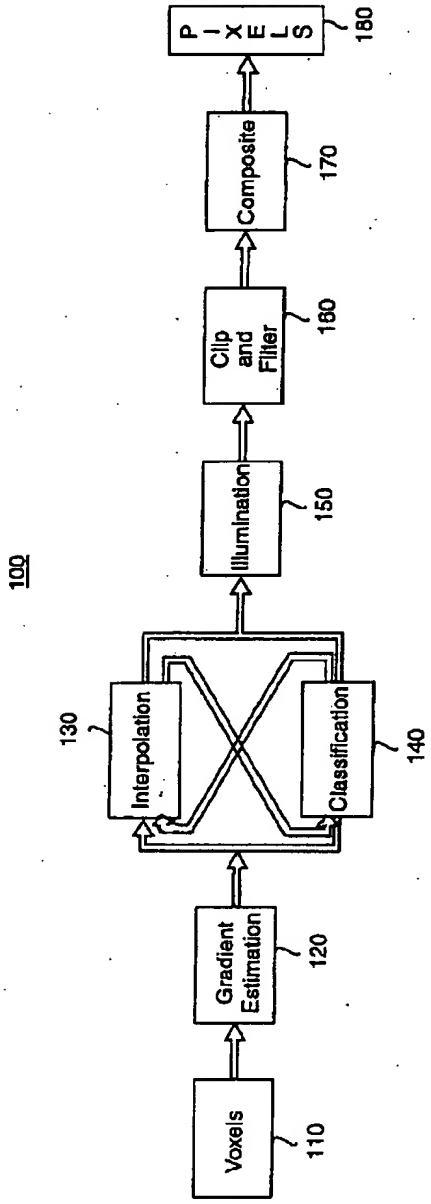

100
Voxels
110
Gradient Estimation
120
Interpolation
130
Classification
140
Illumination
150
Clip and Filter
160
Composite
170
PIXELS
180

FIG. 1

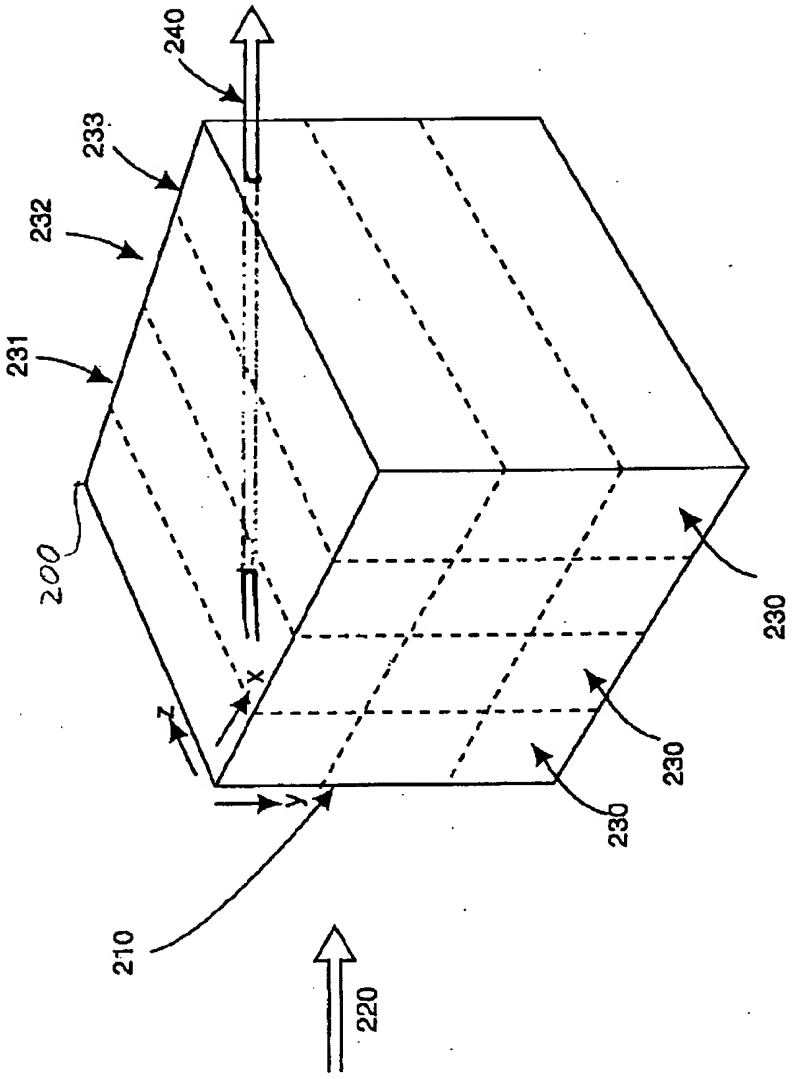


FIG. 2

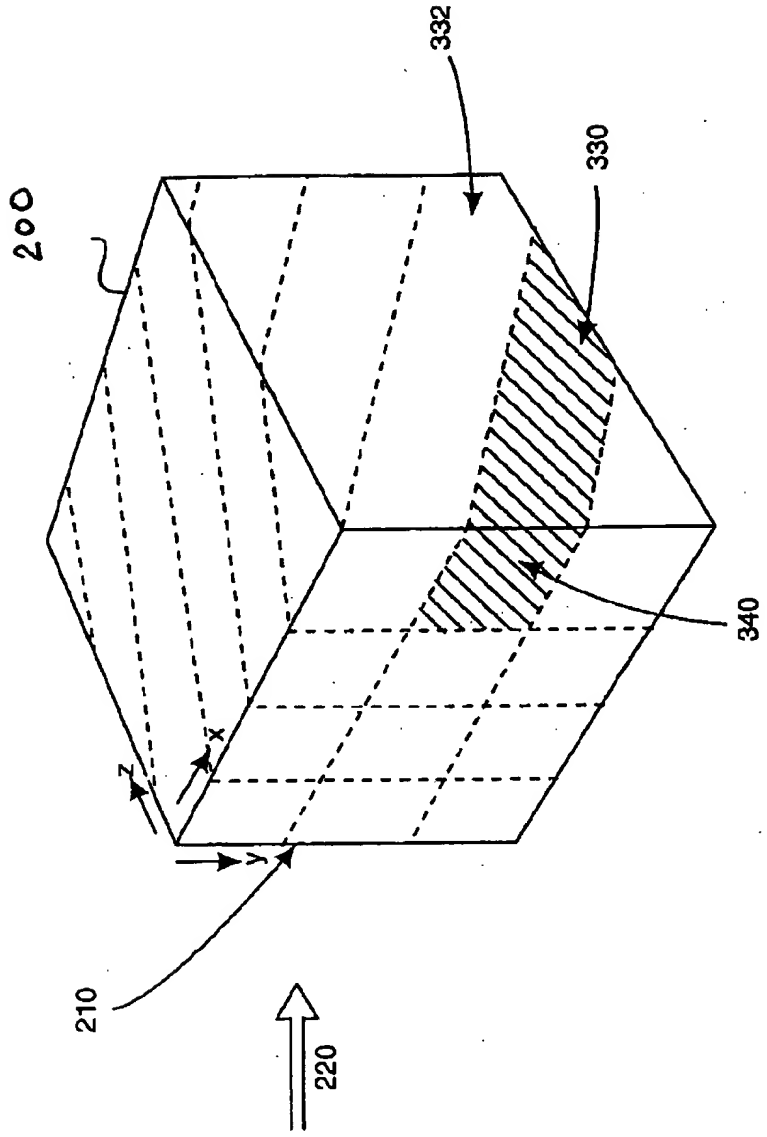


FIG. 3

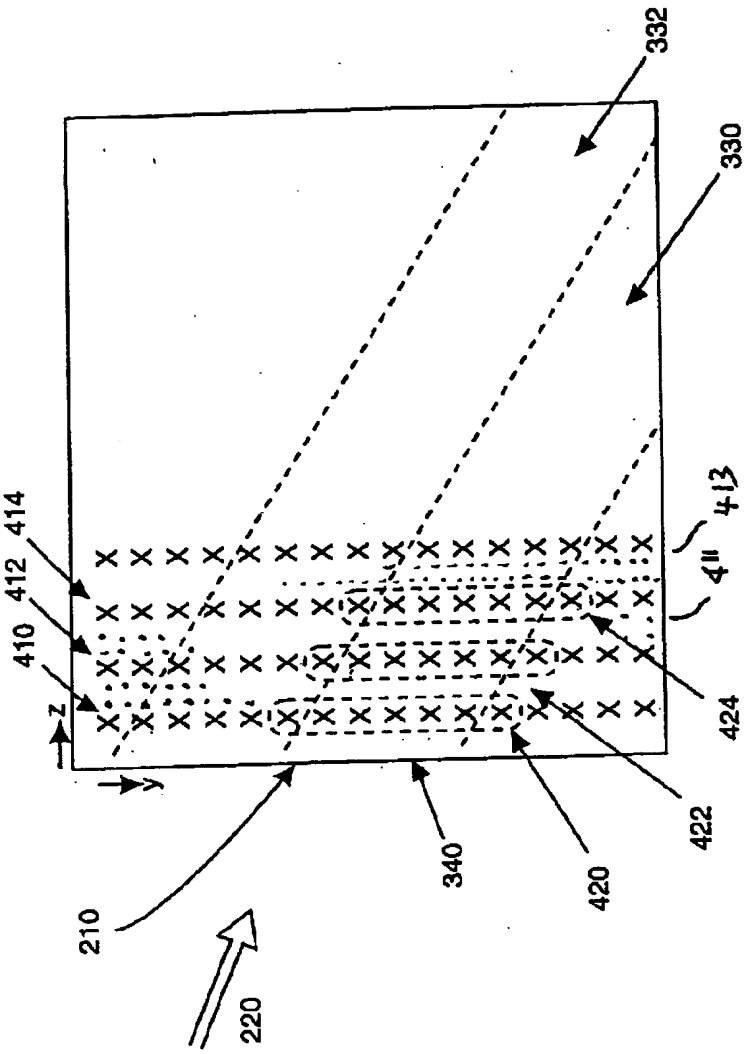


FIG. 4

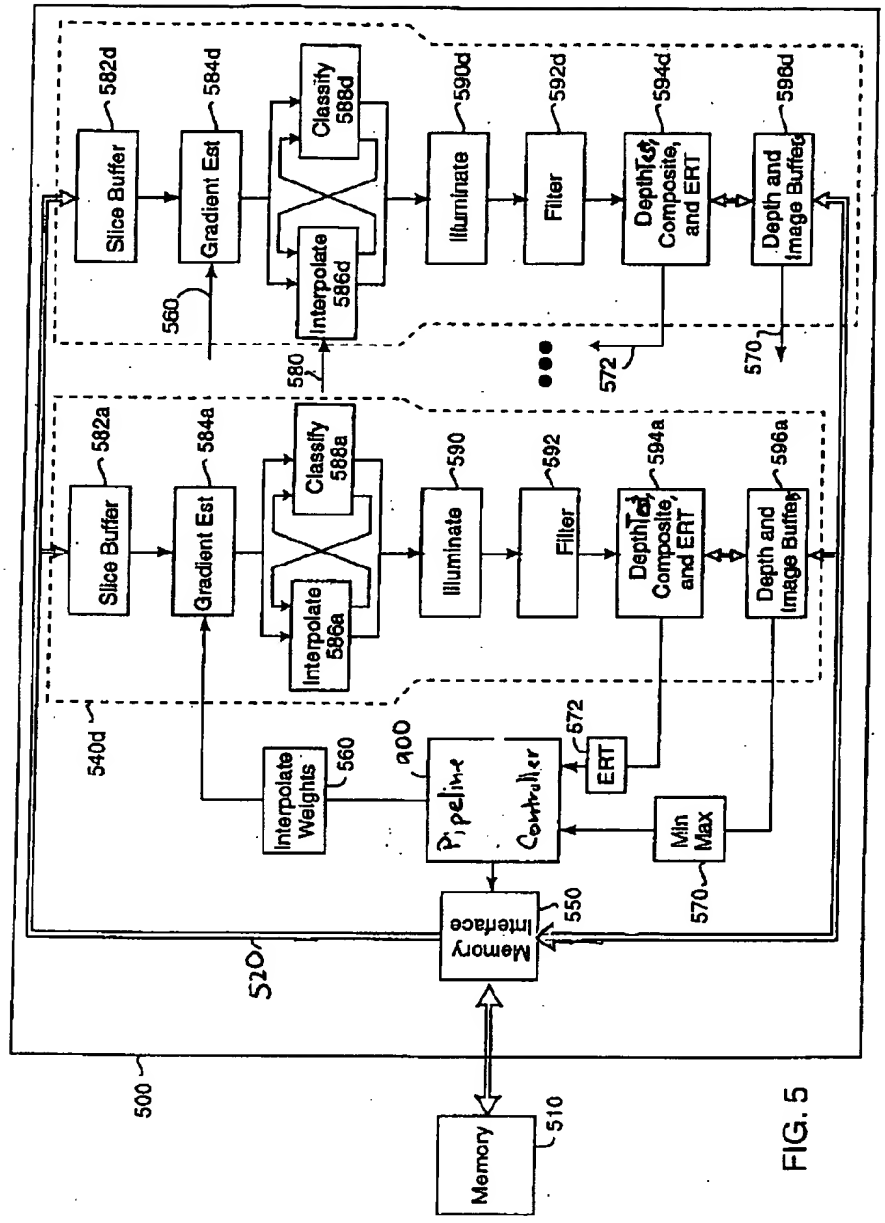

582d
584d
Classify 588d
590d
592d
594d
596d
Slice Buffer
Gradient Est
Interpolate 586d
Illuminate
Filter
Depth, Composite, and ERT
Depth and Image Buffer
560
580
572
570
582a
584a
Classify 588a
590
592
594a
596a
Slice Buffer
Gradient Est
Interpolate 586a
Illuminate
Filter
Depth, Composite, and ERT
Depth and Image Buffer
540d
Interpolate Weights
560
400
Pipeline Controller
572
ERT
Min Max
570
Memory Interface
550
520
500
Memory
510

FIG. 5

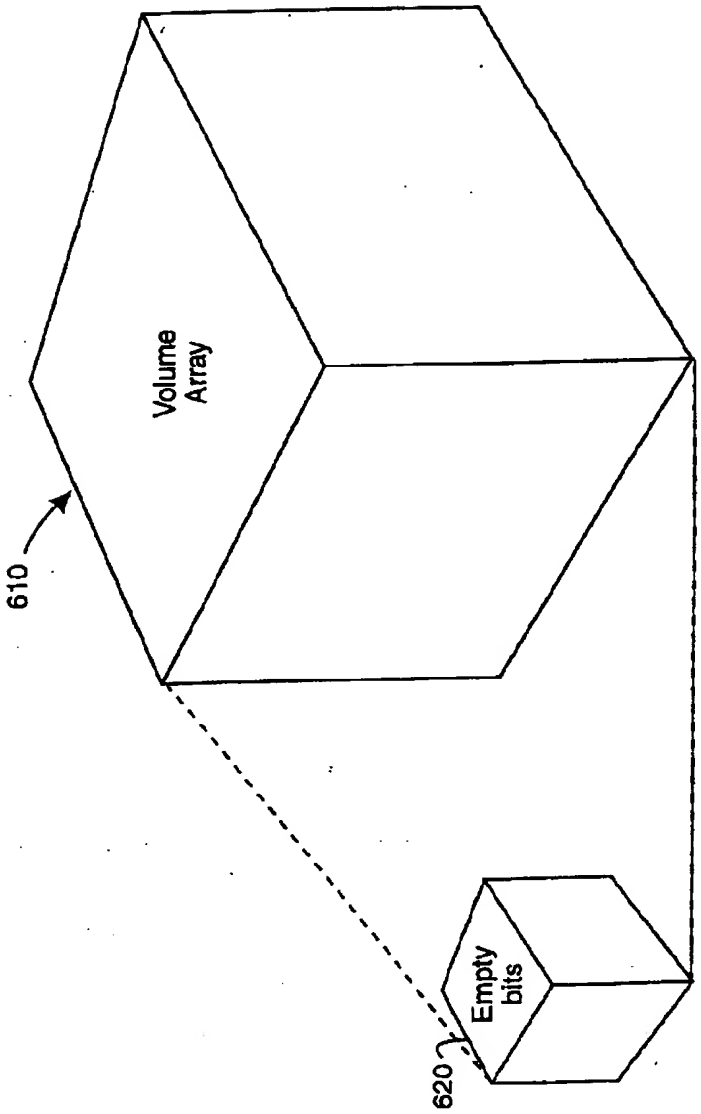


FIG. 6

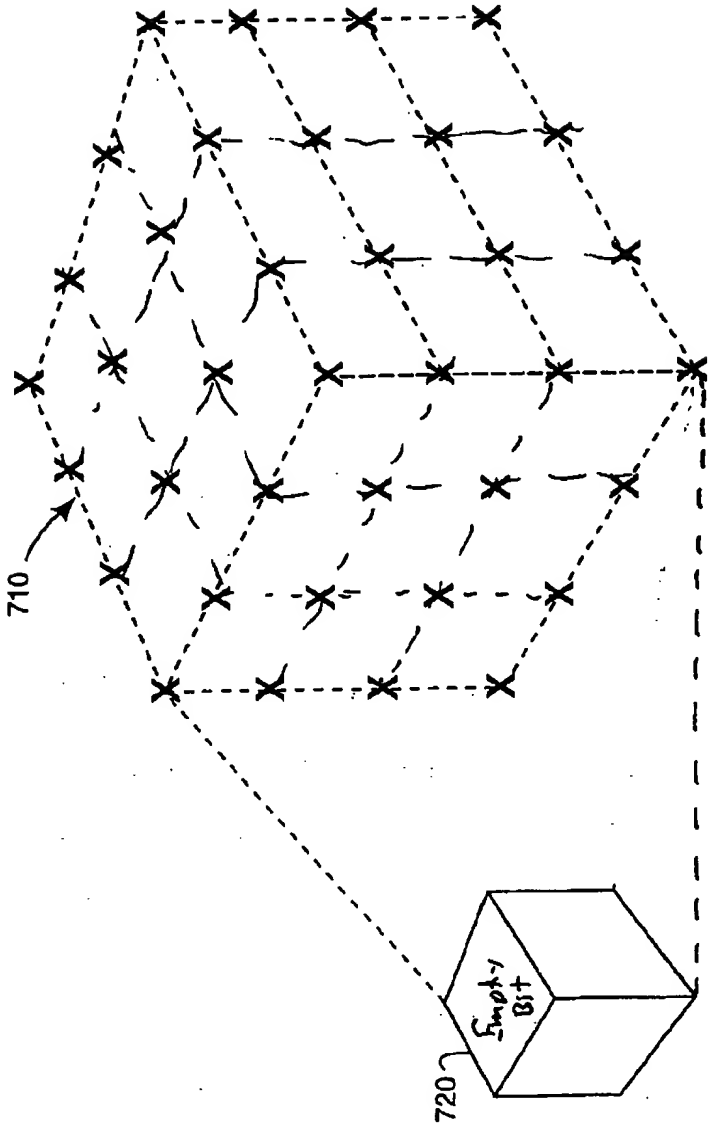


FIG. 7

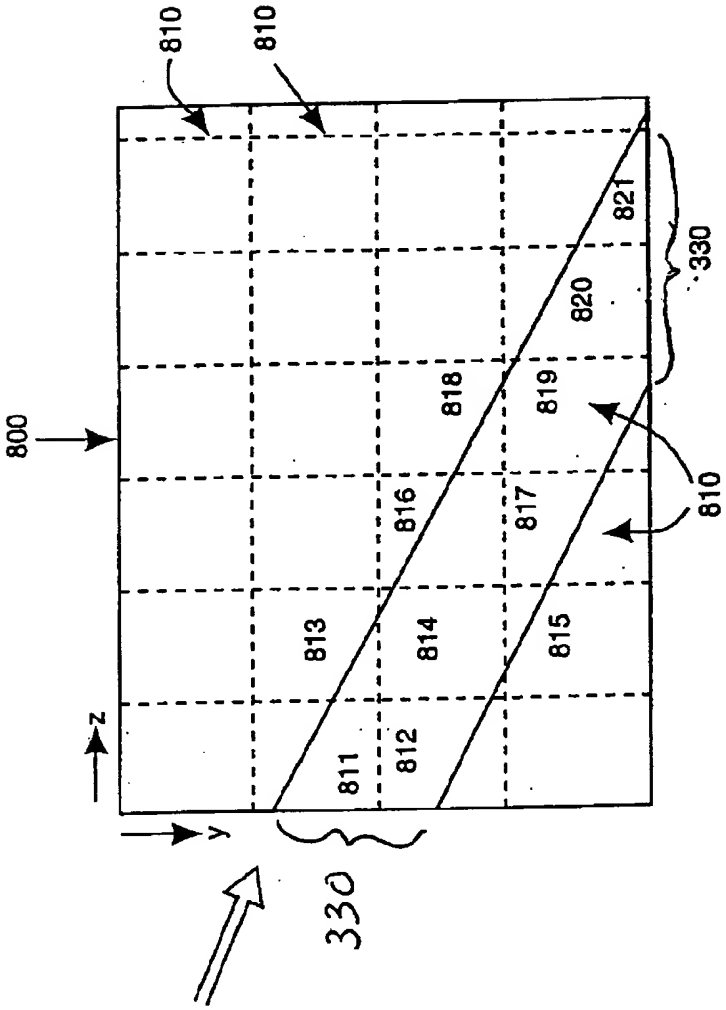

810
810
821
330
820
818
819
800
816
817
810
813
814
815
z
811
812
y
330

FIG. 8

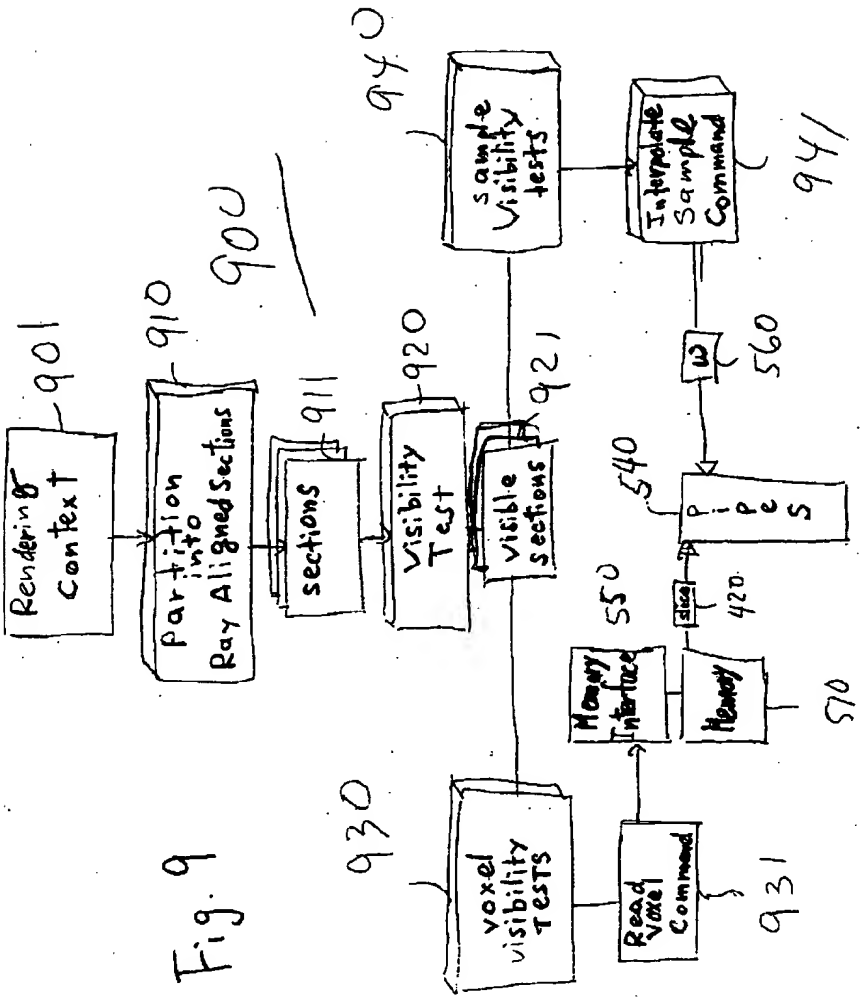

900
901
Rendering Context
910
Partition into Ray Aligned Sections
911
sections
920
Visibility Test
921
visible sections
930
voxel visibility tests
931
Read Voxel Command
940
sample Visibility tests
941
Interpolate Sample Command
560
540
Pipes
550
Memory Interface
420
Memory
570

Fig. 9

1 Abstract

A method renders a volume data set as an image in a volume rendering system. The volume data set includes a plurality of voxels stored in a memory. The volume rendering system includes a plurality of parallel processing pipelines. The image includes a plurality of pixels stored in the memory. A set of rays are cast through the volume data set. The volume data set is partitioned into a plurality of sections aligned with the sets of rays. Voxels along each ray of each set are sequentially interpolated voxels in only one of the plurality of pipelines to generate samples only as long as the samples contribute to the image.

2 Representative Drawing
Fig. 4